

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 1 Metaheurísticas basadas en trayectorias**

3º Grado en Ingeniería Informática—Metaheurística.

## Índice

Esquema de representación.....	3
Calculo de distancias .....	3
Factorización .....	3
Generación de vecinos .....	4
Función de evaluación.....	4
Búsqueda Local.....	5
Pseudocódigo .....	5
Búsqueda Tabú.....	6
Pseudocódigo .....	6
Manejo de la lista tabú.....	7
<i>Pseudocódigo</i> .....	7
Mecanismos de reinicialización.....	7
<i>Pseudocódigo aleatorio</i> .....	7
<i>Pseudocódigo del mejor</i> .....	7
<i>Pseudocódigo menos visitado</i> .....	7
Búsqueda GRASP .....	8
Pseudocódigo .....	8
Greedy aleatorio.....	8
K-Medias.....	9
Manual de usuario .....	9
Parámetros.....	12
Patrones .....	12
Dimensiones de cada patrón.....	12
Número de cluster.....	12
Semillas .....	12
Resultados obtenidos.....	12
Búsqueda Local.....	12
Búsqueda Tabú.....	13
GRASP .....	13
Kmedias .....	14
Análisis de los resultados obtenidos .....	14

# 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

Aunque cada algoritmo tenga sus peculiaridades y busque optimizar la solución de una manera distinta, todos tienen cosas en común, que se describen en los siguientes apartados.

## Esquema de representación.

Para cargar los patrones en el programa utilizamos un vector de punteros a doubles, los cuales serán los patrones. Igual para los centroides de cada cluster, en la posición  $i$  se guardara el centroide del Cluster  $i$  (A partir de ahora  $C_i$ )

Una solución sería la asignación de los  $M$  patrones a los  $K$  cluster por lo tanto usaremos una representación de orden que será un vector de tamaño  $M$  y que en la posición  $i$  guardara el cluster que pertenece el patrón  $i$

## 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  $x_{1\dots n}$  las componentes del patrón  $X$  e  $y_{1\dots n}$  las componentes del patrón  $Y$ .

## Factorización

Cada vez que se desea mover un patrón de un cluster a otro, calcular de nuevo todos los centroides y la función de evaluación, es muy ineficiente, por eso nos ayudamos de la factorización.

Esta funcionalidad tiene en cuenta tanto el incremento/decremento del valor de la solución actual, como de la nueva posición de los centroides.

Empezando con el valor de la solución, esta factorización se resuelve con la siguiente formula:

$$\frac{-m_i}{m_i - 1} * ||P - Z_i||^2 + \frac{m_j}{m_j + 1} * ||P - Z_j||^2$$

Dónde:

$Z_i$  es el cluster que pierde el patrón

$Z_j$  es el cluster que gana el patrón

$m$  es el tamaño del cluster (con  $i \neq j$ )

P es el patrón que movemos

Z el centroide del cluster (con  $i \neq j$ )

Y la factorización del centroide, calculado con:

$$Z_i = \frac{m_i * Z_i - P}{m_i - 1}; Z_j = \frac{m_j * Z_j - P}{m_j - 1}$$

## Generación de vecinos

Los vecinos se generan a partir del movimiento de un patrón de un cluster, hacia otro cluster.

El funcionamiento sería el siguiente.

Cambiar el número de cluster en la posición del patrón dentro del vector de solución.

## Función de evaluación

Nos dice cuan de buena es nuestra solución. Se basa en:

$$\sum ||X^i - Z^j||^2$$

Donde  $X^i$  es cada uno de los patrones del cluster j

Y  $Z^j$  es el centroide de dicho cluster j

# Descripción de cada algoritmo

## Búsqueda Local

La búsqueda local es un proceso de optimización en el que para cada patrón de cada cluster, 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

```
Mientras (contador<10000 && Haya Cambios){  
    Cambios=false  
    Para cada patron i{  
        Para cada Cluster  
            Si (Cj distinto de Ci){  
                Si (incrementoJ(elementoI, Ci, Cj)){  
                    Factorizar  
                    Cambios=true;  
                    Cambiar cluster  
                    Recalcular Centroides  
                }  
            }  
        }  
    }  
}
```

## Búsqueda Tabú

La búsqueda Tabú se basa en buscar 20 movimientos aleatorios de un patron a un cluster distinto al suyo, y comprobar que el mejor movimiento no esté en la lista tabu. En el caso de que esté en la lista tabú solo se aceptara si pasa el criterio de aceptación que en nuestro caso es si mejora la solución.

Cada 2000 iteraciones, haremos un cambio grande en la solución del problema. Nuestra condición de parada serán las 10000 iteraciones.

## Pseudocódigo

```

Mientras contador < 10000{
    Para cada cluster k con tamaño mayor 20 {
        Mientras contador2<20{
            Seleccionar un patron aleatorio
            Seleccionar un cluster distinto aleatorio
            Aleatorio de 0-99
            Si aleatorio<5{
                Contador2++
                Si (Moviento es el mejor de los 20)
                    Guardar movimiento
            }
        }
        Comprobamos si el mejor mov esta o no en la ListaTabu
        Si esta y supera el critero de aspiración o no esta.{
            Actualizamos la lista tabu
            Factorizamos
            Realizamos el moviento.
            Actualizamos la matriz de frecuencias
            Contador++
            ContadorReinicio++
        }
        Si solución es mejor que la MejorSolucion la guardamos.
        Si contadorReinicio++){
            contadorReinicio=0;
            Alatorio de 0-99
            Si aleatorio <25 Reinicio Aleatorio
            Si aleatorio entre 25 y 50 Reinicio mejor solución
            Si aleatorio >=50 Reinicio por menos visitado
            Aleatorio de 0-50
            Si aleatorio< 50
                Aumentamos el tamaño de la ListaTabu un 25%
            Si no
                Disminuimos el tamaño de la ListaTabu un 25%
        }
    }
}

```

## Manejo de la lista tabú

Se basa en una cola FIFO implementada con una lista donde se puede sacar y meter, por arriba y por abajo. Así que a cada movimiento aceptado, primero miro si las dimensiones de esta lista igualan el límite, si es así saco el primero y después introduzco el movimiento Tabu al final.

Para nosotros un movimiento Tabu es un par  $(i, C_i)$  donde  $i$  es el número del patrón y  $C_i$  el cluster al que pertenece ese patrón.

### *Pseudocódigo*

```
Si dimensión lista = limite listaTabu
    Sacar por arriba
Fin Si
Introducir al final movimientoTabu
```

## Mecanismos de reinicialización

Cada 2000 iteraciones, hay que hacer unos cambios grandes en el programa para que no se quede en mínimos locales y así poder encontrar una mejor solución.

### *Pseudocódigo aleatorio*

```
Limpiar lista actual
Para cada patrón
    Aleatorio
    Introducir el patrón en el cluster aleatorio
Fin Para
Calcular centroides
```

### *Pseudocódigo del mejor*

```
Recuperar vector de pertenencias al cada cluster
Actualizar valor de la función de evaluación
Calcular centroide
```

### *Pseudocódigo menos visitado*

```
Para cada patrón i
    Para cada cluster j
        Si Matriz Frecuencias[i][j]<menor
            Menor= matriz Frecuencias[i][j]
            Seleccionar Cluster j
        Fin Si
    Fin Para
    Aleatorio
    Introducir patrón en cluster seleccionado
Fin Para
Calculo de centroides
```



## Búsqueda GRASP

La búsqueda GRASP se apoya en la búsqueda local para poder optimizar sus soluciones, pero al contrario que la búsqueda local, este no construye sus soluciones de forma aleatoria, sino que utiliza un algoritmo de Kauffman aleatorizado para conseguir otros resultados distintos que más tarde serán optimizados.

### Pseudocódigo

Limpiar solución actual  
Hacer un greedy aleatorio  
Búsqueda Local

### Greedy aleatorio

Suma de todos los componentes de todos los patrones y dividir entre el número de patrones

Buscar el patrón más cercano a la suma anterior

Asignar como primer cluster

Para cada cluster menos el primero

    Para cada patrón i

        Si ya esta seleccionado como cluster

            Ganancia[i] infinito

        Si no

            Ganancia[i]=0

            Para cada patrón j

                Si no seleccionado como cluster e  $i \neq j$

                    Para cada cluster seleccionado

                        Dist=Buscar cluster mas cercano

                    Fin Para

                    Valor=dist- distancia(Patron i, Patro j)

                    Si valor>0

                        Sumar ganancia[i]+ valor

                    Fin Si

                Fin Si

            Fin para

        Fin Si

    Buscar ganancia máxima y mínima

    Para cada patrón

        Si su ganancia es menor a  $máximo - 0.3 * (máximo - mínimo)$

            Meter en lista LRC

        Fin Si

    Fin Para

    Aleatorio

    Seleccionar como nuevo cluster LRC[Aleatorio]

    Limpiar lista LRC

Fin Para

Fin Para

## K-Medias

K-Medias es un proceso que empieza con una asignación aleatoria de que patrones pertenecen a que cluster.

Repetir lo siguiente:

Se saca una media de todos los patrones que pertenecen a ese cluster y ese es el centroide del cluster.

Y se va mirando la cercanía de un patrón en el cluster  $i$ , al resto de cluster, al más cercano, se cambia el patrón del cluster  $i$  al cluster más cercano  $j$ .

Y esto se repite hasta que ningún patrón cambie de cluster.

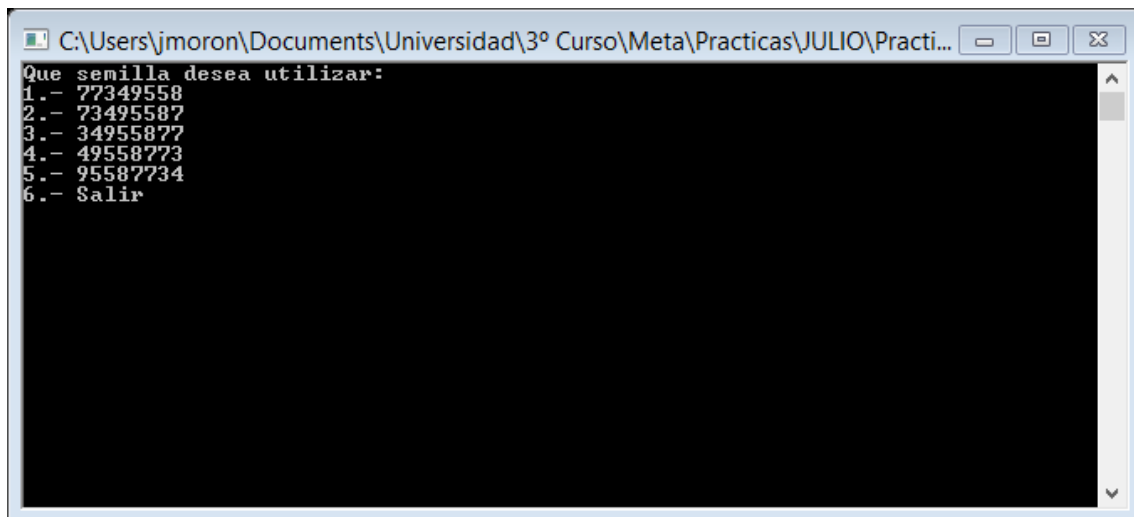
# Procedimiento para el desarrollo de la práctica

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

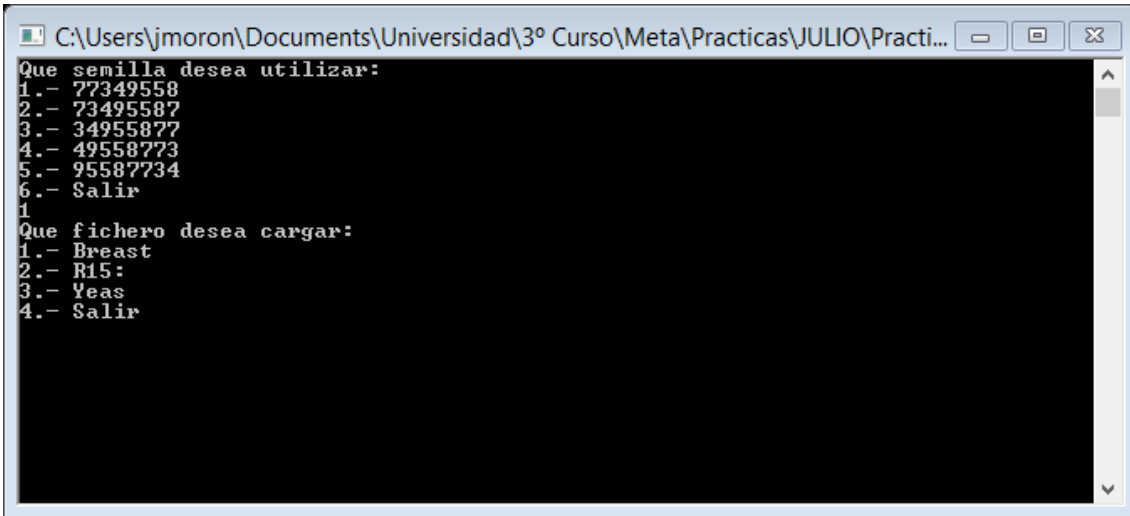
Lo primero que hay que hacer, es meter los ficheros dentro de la misma carpeta donde está el ejecutable de la aplicación.

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



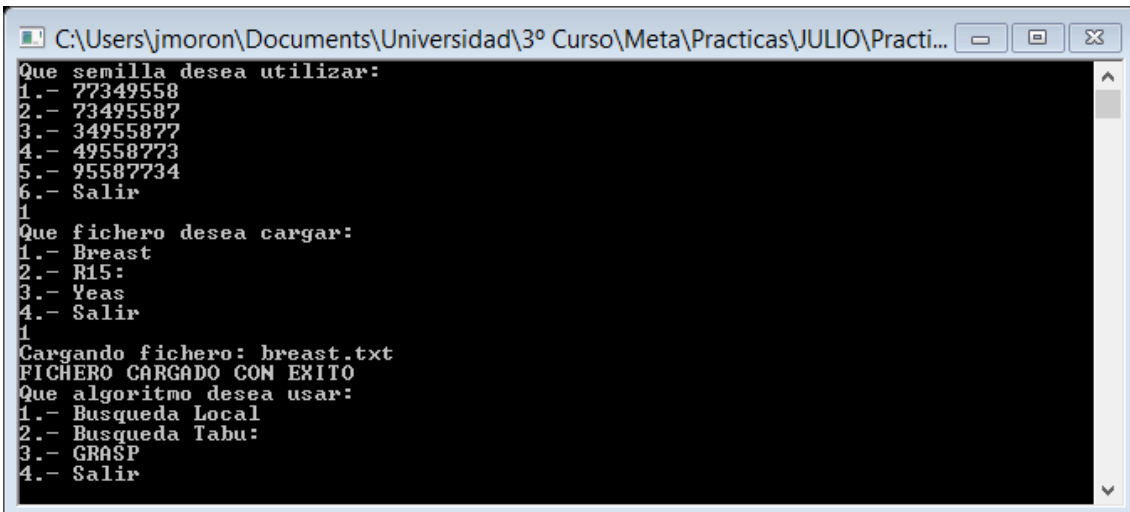
```
C:\Users\jmoron\Documents\Universidad\3º Curso\Meta\Practicas\JULIO\Practi...
Que semilla desea utilizar:
1.- 77349558
2.- 73495587
3.- 34955877
4.- 49558773
5.- 95587734
6.- Salir
```

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



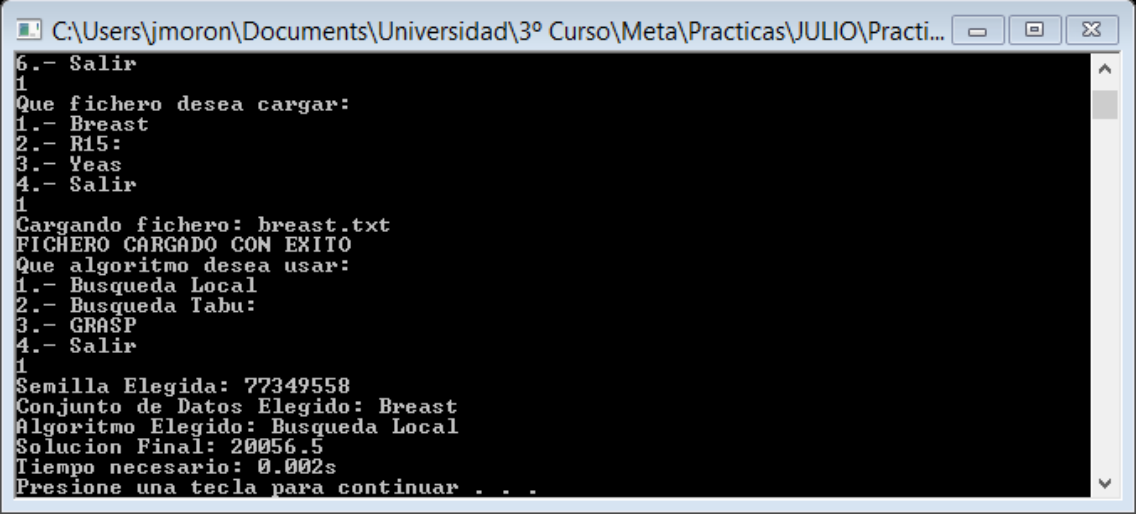
```
C:\Users\jmoron\Documents\Universidad\3º Curso\Meta\Practicas\JULIO\Practi...
Que semilla desea utilizar:
1.- 77349558
2.- 73495587
3.- 34955877
4.- 49558773
5.- 95587734
6.- Salir
1
Que fichero desea cargar:
1.- Breast
2.- R15:
3.- Yeas
4.- Salir
```

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



```
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
1
Cargando fichero: breast.txt
FICHERO CARGADO CON EXITO
Que algoritmo desea usar:
1.- Busqueda Local
2.- Busqueda Tabu:
3.- GRASP
4.- Salir
```

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



```
C:\Users\jmoron\Documents\Universidad\3º Curso\Meta\Practicas\JULIO\Practi...
6.- Salir
1
Que fichero desea cargar:
1.- Breast
2.- R15:
3.- Yeas
4.- Salir
1
Cargando fichero: breast.txt
FICHERO CARGADO CON EXITO
Que algoritmo desea usar:
1.- Busqueda Local
2.- Busqueda Tabu:
3.- GRASP
4.- Salir
1
Semilla Elegida: 77349558
Conjunto de Datos Elegido: Breast
Algoritmo Elegido: Busqueda Local
Solucion Final: 20056.5
Tiempo necesario: 0.002s
Presione una tecla para continuar . . .
```

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

### Búsqueda Local

	Breast		R15		Yeast	
	<i>J</i>	Tiempo	<i>J</i>	Tiempo	<i>J</i>	Tiempo
<b>Ejecución 1</b>	20056,50	0,002	284,27	0,03	46,96	0,04
<b>Ejecución 2</b>	20258,10	0,003	6740,11	0,01	66,69	0,02
<b>Ejecución 3</b>	19720,00	0,003	error	error	46,74	0,04
<b>Ejecución 4</b>	20171,00	0,003	990,16	0,15	46,82	0,06
<b>Ejecución 5</b>	19720,00	0,002	298,48	0,04	48,05	0,07
<b>Mejor</b>	19720,00	---	284,27	---	46,74	---
<b>Peor</b>	20258,10	---	6740,11	---	66,69	---
<b>Media</b>	19985,12	0,003	2078,25	0,06	51,05	0,05
<b>Desv. típica</b>	252,36	0,001	3125,32	0,06	8,76	0,02

## Búsqueda Tabú

	Breast		R15		Yeast	
	<i>J</i>	Tiempo	<i>J</i>	Tiempo	<i>J</i>	Tiempo
Ejecución 1	19716,30	27,600	1776,38	10,62	60,78	13,37
Ejecución 2	19716,30	474,901	1895,08	8,64	63,35	12,73
Ejecución 3	19716,30	77,969	1874,52	10,49	60,50	18,77
Ejecución 4	19716,30	23,546	1918,32	9,99	60,28	13,53
Ejecución 5	19716,30	17,665	1749,64	8,56	60,42	18,04
Mejor	19716,30	---	1749,64	---	60,28	---
Peor	19716,30	---	1918,32	---	63,35	---
Media	19716,30	124,336	1842,79	9,66	61,07	15,29
Desv. típica	0,00	197,447	75,06	1,00	1,29	2,87

## GRASP

	Breast		R15		Yeast	
	<i>J</i>	Tiempo	<i>J</i>	Tiempo	<i>J</i>	Tiempo
Ejecución 1	19716,80	5,252	108,62	158,65	47,35	550,01
Ejecución 2	19716,80	5,265	108,62	159,05	48,14	549,44
Ejecución 3	19716,80	5,276	108,62	159,43	47,23	556,84
Ejecución 4	19716,80	5,247	108,62	158,31	46,83	623,90
Ejecución 5	19716,80	5,320	108,62	159,29	47,46	569,82
Mejor	19716,80	---	108,62	---	46,83	---
Peor	19716,80	---	108,62	---	48,14	---
Media	19716,80	5,272	108,62	158,95	47,40	570,00
Desv. típica	0,00	0,029	0,00	0,46	0,48	31,23

## 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

	Breast		R15		Yeast	
	<i>J</i>	Tiempo	<i>J</i>	Tiempo	<i>J</i>	Tiempo
BL (med)	19985,12	0,00	2078,26	0,06	51,05	0,05
BL (desv)	252,36	0,00	3125,32	0,06	8,76	0,02
BT	19716,30	124,34	1842,79	9,66	61,07	15,29
	0,00	197,45	75,06	1,00	1,29	2,87
GRASP	19716,80	5,27	108,62	158,95	47,40	570,00
	0,00	0,03	0,00	0,46	0,48	31,23
KM	19716,30	0,01	1355,95	0,03	46,03	0,15
	0,00	0,00	584,36	0,01	0,26	0,04