



## PRÁCTICA 3

*Sistema de Colonias de Hormigas*

Autores:

David Díaz Jiménez 77356084T

Andrés Rojas Ortega 77382127F

Grupo 2

# Metaheurísticas

## Informe de prácticas

David Díaz Jiménez, Andrés Rojas Ortega

### Contents

<b>1</b>	<b>Definición y análisis del problema</b>	<b>2</b>
1.1	Representación de la solución . . . . .	2
1.2	Función objetivo . . . . .	2
1.3	Operadores comunes . . . . .	2
<b>2</b>	<b>Clases auxiliares</b>	<b>2</b>
2.1	Archivo . . . . .	2
2.2	Configurador . . . . .	3
2.3	GestorLog . . . . .	3
2.4	Metaheurísticas . . . . .	3
2.5	Pair . . . . .	3
2.6	RandomP . . . . .	3
2.7	Timer . . . . .	3
<b>3</b>	<b>Pseudocódigo</b>	<b>4</b>
3.1	algoritmo principal . . . . .	4
3.2	Inicialización de la matriz de feromona . . . . .	5
3.3	Inicialización de la colonia de hormigas . . . . .	5
3.4	Generación de la Lista Restringida de Candidatos . . . . .	6
<b>4</b>	<b>Experimentos y análisis de resultados</b>	<b>8</b>
4.1	Procedimiento de desarrollo de la práctica . . . . .	8
4.1.1	Equipo de pruebas . . . . .	8
4.1.2	Manual de usuario . . . . .	9
4.2	Parámetros de los algoritmos . . . . .	9
4.2.1	Sistema de Colonias de Hormigas . . . . .	9
4.2.2	Semillas . . . . .	9
4.3	Análisis de los resultados . . . . .	9

## 1 Definición y análisis del problema

Dado un conjunto  $N$  de tamaño  $n$ , se pide encontrar un subconjunto  $M$  de tamaño  $m$ , que maximice la función:

$$C(M) = \sum_{s_i, s_j \in M} d_{ij}$$

donde  $d_{ij}$  es la diversidad del elemento  $s_i$  respecto al elemento  $s_j$

### 1.1 Representación de la solución

Para representar la solución se ha optado por el uso de un vector de enteros, en el que el elemento contenido en cada posición se corresponde con un integrante de la solución. La solución vendrá dada por las siguientes restricciones:

- La solución no puede contener elementos repetidos.
- Debe tener exactamente  $m$  elementos.
- El orden de los elementos es irrelevante.

### 1.2 Función objetivo

$$C(M) = \sum_{s_i, s_j \in M} d_{ij}$$

### 1.3 Operadores comunes

El operador de intercambio es el 1-opt, se seleccionara un elemento de la solución actual en base a un criterio y se sustituirá por un elemento que no pertenece a la solución.

## 2 Clases auxiliares

A continuación se enumeran las diferentes clases auxiliares utilizadas en el programa acompañadas de una breve descripción de las mismas.

**nota:** Para obtener información detallada se deben consultar los comentarios insertados en el código de cada una de las clases.

### 2.1 Archivo

Esta clase se encarga de almacenar toda la información que se encuentra dentro de cada archivo que contiene cada uno de los problemas.

## **2.2 Configurador**

Utilizamos esta clase para leer y almacenar los parámetros del programa que se encuentran dentro del archivo de configuración.

## **2.3 GestorLog**

La función principal de esta clase es la administración de los archivos Log del programa y el almacenamiento de información para debug en los mismos.

## **2.4 Metaheurísticas**

Esta clase se utiliza para lanzar la ejecución de los algoritmos para cada problema facilitado como parámetro.

## **2.5 Pair**

Representa un par formado por un candidato y un coste asociado a este.

## **2.6 RandomP**

Clase para generar números aleatorios.

## **2.7 Timer**

Clase para gestionar los tiempos de ejecución del algoritmo.

## 3 Pseudocódigo

### 3.1 algoritmo principal

---

**Algorithm 1** Sistema Colonia Hormigas

---

```
matrizFeromonas  $\leftarrow$  InicializarFeromona()  
while iteracionesRealizadas  $\leq$  limiteIteraciones  $\wedge$  tiempoEjecucion  $\leq$  limiteTiempoEjecucion do  
    colonia  $\leftarrow$  InicializarColonia()  
    for i=1; i < tamañoHormiga; i++ do  
        for hormiga  $\in$  colonia do  
            lrc  $\leftarrow$  GenerarLRC(hormiga)  
            AplicarReglaTransicion(lrc, hormiga)  
        end for  
        ActualizarFeromonaLocal()  
    end for  
    TareasDemonio()  
     $\emptyset \leftarrow$  colonia  
    iteracionesRealizadas  $\leftarrow$  iteracionesRealizadas + 1  
end while
```

---

Lo primero que realiza el algoritmo es inicializar la matriz de feromonas con el valor inicial. Esta tarea la realiza la función "InicializarFeromona()", la matriz de feromonas se almacena en la variable "matrizFeromonas". El valor inicial de feromona se pasa como parámetro del programa.

A continuación, entramos en un bucle while cuyas condiciones de parada son: las iteraciones realizadas y el tiempo de ejecución del algoritmo. En el caso de las iteraciones realizadas, almacenadas en la variable "iteracionesRealizadas", deben ser inferiores al límite almacenado en la variable "limiteIteraciones". El valor de "limiteIteraciones" se pasa como parámetro del programa. En el caso del tiempo de ejecución, que se almacena en la variable "tiempoEjecucion", debe de ser inferior al valor almacenado en la variable "limiteTiempoEjecucion". El valor de "limiteTiempoEjecucion" se pasa como parámetro del programa.

Cada vez que empieza una nueva iteración del bucle, lo primero que se realiza es inicializar la colonia de hormigas. La colonia se almacena en la variable "colonia" y la función encargada de inicializarla es "InicializarColonia()".

Una vez inicializada la colonia, iniciamos dos bucles for. El primero se ejecutará hasta que el valor de "i" alcance el número de elementos que debe contener cualquier solución (parámetro del programa, almacenado en la variable "tamañoHormiga"). El segundo bucle for recorrerá cada una de las hormigas pertenecientes a la variable "colonia".

Lo primero que realizamos es almacenar en la variable "lrc" la lista de candidatos restringida de la hormiga, calculada por la función "GenerarLRC(hormiga)".

Una vez almacenada la lista restringida, seleccionamos uno de los candidatos que contiene aplicando la regla de la transición y lo añadimos como parte de la solución de la hormiga. La función que realiza dicha tarea es "AplicarReglaTransicion(lrc,hormiga)".

Se finaliza el bucle for interior y la función "ActualizarFeromonaLocal()" se encarga de actualizar la feromona local de todas las hormigas después de haber añadido un nuevo elemento.

Una vez finalice el bucle for tendremos todas las hormigas completas, por lo que pasamos a seleccionar la mejor hormiga de la colonia y realizar la actualización de la feromona global. La función encargada de dichas tareas es "TareasDemonio()".

Para finalizar el bucle while, vaciamos la variable "colonia" y actualizamos el valor de la variable "iteracionesRealizadas".

### 3.2 Inicialización de la matriz de feromona

**Salida** Devuelve la matriz de feromonas inicializada.

---

**Algorithm 2** Inicialización de la matriz de feromona

---

```

for i=0; i<tamañoMatriz; i++ do
  for j=0; j<tamañoMatriz; j++ do
    matrizFeromonas[i][j] ← feromonaInicial
  end for
end for
return matrizFeromonas

```

---

El funcionamiento de esta función es trivial: se inicializan todos los elementos de la matriz de feromonas con el valor de "feromonaInicial" (parámetro del programa).

### 3.3 Inicialización de la colonia de hormigas

**Salida** Devuelve la colonia de hormigas inicializada.

---

**Algorithm 3** Inicialización de la colonia de hormigas

---

```
while colonia.tamaño() < tamañoColonia do  
    hormiga  $\leftarrow \emptyset$   
    primerElemento  $\leftarrow \text{GenerarEnteroAleatorio}()$   
    hormiga  $\leftarrow \text{hormiga} \cup \{\text{primerElemento}\}$   
    colonia  $\leftarrow \text{colonia} \cup \{\text{hormiga}\}$   
end while  
return colonia
```

---

El funcionamiento de esta función es trivial: se inicializan todas las hormigas con un elemento generado aleatoriamente con la función "GenerarEnteroAleatorio()". Cuando se tengan "tamañoColonia" (parámetro del programa) hormigas inicializadas, se devuelve la colonia como resultado.

### 3.4 Generación de la Lista Restringida de Candidatos

**Entrada** La hormiga para la que se quiere obtener su Lista Restringida de Candidatos.

**Salida** Devuelve la Lista Restringida de Candidatos "lrc".

---

**Algorithm 4** Generación de la Lista Restringida de Candidatos

---

```
lrc  $\leftarrow \emptyset$ 
noSeleccionados  $\leftarrow \emptyset$ 
min  $\leftarrow +\infty$ 
max  $\leftarrow -\infty$ 
for i=0;i<tamañoMatriz;i++ do
  if i  $\notin$  hormiga then
    aporte  $\leftarrow 0$ 
    for elemento  $\in$  hormiga do
      aporte  $\leftarrow$  aporte + MatrizCostes[elemento][i]
    end for
    if min ==  $+\infty$  then
      max  $\leftarrow$  aporte
      min  $\leftarrow$  aporte
    end if
    if aporte > max then
      max  $\leftarrow$  aporte
    else if aporte < min then
      min  $\leftarrow$  aporte
    end if
    noSeleccionados  $\leftarrow$  noSeleccionados  $\cup \{i, aporte\}$ 
  end if
end for
valorCorte  $\leftarrow$  min + delta * (max - min)
for elemento  $\in$  noSeleccionados do
  if elemento.aporte  $\geq$  valorCorte then
    lrc  $\leftarrow$  lrc  $\cup \{elemento\}$ 
  end if
end for
return lrc
```

---

Lo primero que se realiza es inicializar los valores de las variables "lrc", "noSeleccionados", "min" y "max". "lrc" almacena la Lista Restringida de Candidatos, "noSeleccionados" almacena todos los elementos del problema que no forman parte de la hormiga, "min" y "max" almacena el mínimo y máximo aporte encontrado hasta el momento.

Se recorren todos los elementos del problema y se comprueba que no se encuentren ya en la hormiga.

En el caso de que no se encuentren en la hormiga, se actualiza el valor de la variable "aporte" a cero y se calcula el coste que aportaría a la hormiga si lo incluyéramos en ella. Este coste se almacena en la variable "aporte".



A continuación, se comprueba que aporte no sea mayor que "max" o menor que "min". En el caso de que se cumpla una de estas dos condiciones, se actualiza el valor de "max" o de "min" dependiendo del caso.

Para finalizar, se añade el elemento junto su aporte a la variable "noSeleccionados".

Se realiza lo anterior hasta que se haya recorrido todos los elementos del problema.

Lo siguiente que se realiza es el cálculo de "valorCorte" aplicando la siguiente fórmula, siendo "delta" un parámetro del programa:

$$valorCorte = min + delta * (max - min)$$

Para cada elemento de "noSeleccionados", se comprueba que su aporte sea mayor que "valorCorte". Si se da este caso, se añade el elemento a la Lista Restringida de Candidatos. Cuando se haya hecho la comprobación de todos los elementos "lrc" habrá sido generada.

Una vez se haya realizado esto último, se devuelve la Lista Restringida de Candidatos generada como resultado de la ejecución de la función.

## 4 Experimentos y análisis de resultados

### 4.1 Procedimiento de desarrollo de la práctica

Para realizar la práctica, se ha optado por implementar las heurísticas propuestas en el lenguaje de programación JAVA. El ejecutable que se entrega junto a este documento ha sido compilado bajo `APACHE NETBEANSIDE 12.0`.

#### 4.1.1 Equipo de pruebas

Los resultados de las heurísticas han sido obtenidos en el siguiente equipo:

- Host:
- S.O:
- Kernel:
- CPU:
- GPU:
- GPU:
- Memoria RAM :

### 4.1.2 Manual de usuario

Para ejecutar el software asegúrese de que el archivo .jar proporcionado se ubica en el mismo directorio que la carpeta *archivos*.

Cuando se muestre la GUI, podrá seleccionar la heurística que desee mediante el botón correspondiente. Una vez empiece la ejecución de una heurística no será posible seleccionar otra hasta que finalice su ejecución. Los resultados finales se mostrarán en el cuadro de texto, a su vez, se generan los log correspondientes a cada archivo y semilla en la carpeta Log.



Figure 1: GUI

## 4.2 Parámetros de los algoritmos

### 4.2.1 Sistema de Colonias de Hormigas

### 4.2.2 Semillas

Para la generación de números pseudoaleatorios se utiliza una semilla previamente definida en el archivo de configuración, en este caso es 77356084. Esta semilla se va rotando en las 5 iteraciones de cada archivo.

77356084  $\rightarrow$  73560847  $\rightarrow$  35608477 ...

## 4.3 Análisis de los resultados

## References

- [1] Umbarkar, Dr. Anantkumar & Sheth, P.. (2015). CROSSOVER OPERATORS IN GENETIC ALGORITHMS: A REVIEW. ICTACT Journal on Soft Computing ( Volume: 6 , Issue: 1 ). 6. 10.21917/ijsc.2015.0150.
- [2] <https://sci2s.ugr.es/graduateCourses/Metaheuristics>
- [3] <https://sci2s.ugr.es/graduateCourses/Algoritmica>