



## PRÁCTICA 2

*Genéticos*

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	ElementoSolucion . . . . .	3
2.4	GestorLog . . . . .	3
2.5	Metaheurísticas . . . . .	3
2.6	Pair . . . . .	3
2.7	RandomP . . . . .	3
2.8	Timer . . . . .	3
<b>3</b>	<b>Pseudocódigo</b>	<b>4</b>
<b>4</b>	<b>Experimentos y análisis de resultados</b>	<b>9</b>
4.1	Procedimiento de desarrollo de la práctica . . . . .	9
4.1.1	Equipo de pruebas . . . . .	9
4.1.2	Manual de usuario . . . . .	9
4.2	Parámetros de los algoritmos . . . . .	10
4.2.1	Genetico . . . . .	10
4.2.2	Semillas . . . . .	10
4.3	Análisis de los resultados . . . . .	10

## 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 ElementoSolucion**

Clase encargada de representar a un elemento perteneciente a una solución y almacenar toda la información necesaria para la ejecución de las metaheurísticas del programa.

## **2.4 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.

## **2.5 Metaheurísticas**

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

## **2.6 Pair**

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

## **2.7 RandomP**

Clase para generar números aleatorios.

## **2.8 Timer**

Clase para gestionar los tiempos de ejecución del algoritmo.

### 3 Pseudocódigo

---

**Algorithm 1** Algoritmo Genético

---

```
poblacionPadres  $\leftarrow$  GeneraPoblacionInicial(semilla)
while evaluacionesRelizadas < evaluacionesLimite do
    Evaluacion(poblacionPadres)
    elites  $\leftarrow$  SeleccionElites(poblacionPadres)
    poblacionHijos  $\leftarrow$  CruzarPoblacion(poblacionPadres, semilla)
    Reparar(poblacionHijos)
    MutarPoblacion(poblacionHijos, semilla)
    Evaluacion(poblacionHijos)
    poblacionPadres  $\leftarrow$  ReemplazarElite(poblacionHijos, elite)
end while
```

---

La primera acción que se realiza en la función principal es la generación de un conjunto de individuos aleatorios que se almacenará en la variable global "poblacionPadres". La función encargada de realizar esta labor es "GeneraPoblacionInicial(semilla)"

---

**Algorithm 2** GeneraPoblacionInicial(semilla)

---

```
individuo  $\leftarrow$   $\emptyset$ 
poblacion  $\leftarrow$   $\emptyset$ 
while tamañoPoblacion < numIndividuosPoblacion do
    while numGenesIndividuo < numGenesIndividuos do
        genAleatorio  $\leftarrow$  GeneraEnteroAleatorio(semilla)
        if genAleatorio  $\notin$  individuo then
            individuo  $\leftarrow$  individuo  $\cup$  {genAleatorio}
        end if
    end while
    poblacion  $\leftarrow$  poblacion  $\cup$  {individuo}
    individuo  $\leftarrow$   $\emptyset$ 
end while
return poblacion
```

---

Inicializamos las variables "individuo" y "poblacion". La variable "individuo" se utilizará como contenedor de todos los genes que se vayan generando aleatoriamente, se inicializa como un conjunto vacío. "poblacion" irá almacenando cada uno de los individuos generados, se inicializa como un conjunto vacío.

Hasta que "poblacion" no contenga el número de individuos especificado como parámetro del programa hacemos lo siguiente:

Generamos un genotipo aleatorio haciendo uso de la función "GeneraEnteroAleatorio(semilla)" y lo almacenamos en la variable local "genAleatorio".

Comprobamos que "genAleatorio" no se encuentre ya contenido dentro de "individuo" y lo añadimos en el caso de que cumpla con esta condición.

Repetimos la generación aleatoria de genotipos hasta que el número de los mismos contenido en "individuo" se corresponda con el número de genotipos pasado como parámetro del programa.

Cuando "individuo" tiene el número de genes deseado, se añade a "poblacion" y acto seguido se modifica el valor de "individuo" a vacío para dar paso a la generación de otro "individuo" nuevo.

Una vez tengamos hayamos completado "poblacion", la devolvemos como resultado de la ejecución de la función.

---

**Algorithm 3** Evaluacion(poblacion)

---



---

**Algorithm 4** SeleccionElites(poblacion)

---

```

individuosElites  $\leftarrow \emptyset$ 
mejor  $\leftarrow \emptyset$ 
costeMejor  $\leftarrow 0$ 
while individuosElites.tamaño() < numElites do
  for individuo  $\in$  poblacion do
    if (individuo.coste > costeMejor)  $\wedge$  (individuo  $\notin$  individuosElite) then
      mejor  $\leftarrow$  individuo
      costeMejor  $\leftarrow$  individuo.coste
    end if
  end for
  individuosElites  $\leftarrow$  individuosElites  $\cup \{mejor\}$ 
end while
return individuosElites

```

---



---

**Algorithm 5** CruzarPoblacion(poblacion,semilla)

---

```

poblacionHijos  $\leftarrow$  SeleccionaPoblacion(poblacion, semilla)
if tipoCruceMPX then
  poblacionHijos  $\leftarrow$  RealizaCruceMPX(poblacionHijos)
else
  poblacionHijos  $\leftarrow$  RelizaCruce2p(poblacionHijos)
end if
return poblacionHijos

```

---

---

**Algorithm 6** SeleccionaPoblacion(poblacion,semilla)

---

**while** tamañoPoblacionHijos<numHijos **do**  
    *individuoSeleccionado*  $\leftarrow$  *SeleccionaIndividuo*(poblacion, semilla)  
    *poblacionHijos*  $\leftarrow$  *poblacionHijos*  $\cup$  {*individuoSeleccionado*}  
**end while**  
**return** poblacionHijos

---

---

**Algorithm 7** SeleccionaIndividuo(poblacion,semilla)

---

*seleccionado1*  $\leftarrow$  *GeneraEnteroAleatorio*(semilla)  
*seleccionado2*  $\leftarrow$  *GeneraEnteroAleatorio*(semilla)  
**while** seleccionado1==seleccionado2 **do**  
    *seleccionado2*  $\leftarrow$  *GeneraEnteroAleatorio*(semilla)  
**end while**  
**if** poblacion[seleccionado1].coste > poblacion[seleccionado2].coste **then**  
    **return** poblacion[seleccionado1]  
**else**  
    **return** poblacion[seleccionado2]  
**end if**

---

---

**Algorithm 8** RealizarCruceMPX(poblacionHijos)

---

---

---

**Algorithm 9** RealizarCruce2p(poblacionHijos)

---

```
for i=0; i<49; i+=2 do
  corte1  $\leftarrow$  GeneraEnteroAleatorio(semilla)
  corte2  $\leftarrow$  GeneraEnteroAleatorio(semilla)
  padre1  $\leftarrow$  poblacionPadre[i]
  padre2  $\leftarrow$  poblacionPadre[i + 1]
  while corte1==corte2 do
    corte2  $\leftarrow$  GeneraEnteroAleatorio(semilla)
  end while
  for j=0;j<corte1;j++ do
    hijo1  $\leftarrow$  hijo1  $\cup$  padre1.getGen[j]
    hijo2  $\leftarrow$  hijo2  $\cup$  padre2.getGen[j]
  end for
  for j=corte1;j<corte2;j++ do
    hijo1  $\leftarrow$  hijo1  $\cup$  padre2.getGen[j]
    hijo2  $\leftarrow$  hijo2  $\cup$  padre1.getGen[j]
  end for
  for j=corte2;j<tamañoIndividuo;j++ do
    hijo1  $\leftarrow$  hijo1  $\cup$  padre1.getGen[j]
    hijo2  $\leftarrow$  hijo2  $\cup$  padre2.getGen[j]
  end for
  poblacionHijos  $\leftarrow$  poblacionHijos  $\cup$  {hijo1}
  poblacionHijos  $\leftarrow$  poblacionHijos  $\cup$  {hijo2}
end for
return poblacionHijos
```

---

---

**Algorithm 10** Reparar(poblacionHijos)

---

```
for individuo  $\in$  poblacionHijos do
  if !FuncionSolucion(individuo) then
    if tamañoIndividuo>tamañoIndividuoProblema then
      while tamañoIndividuo>tamañoIndividuoProblema do
        elementoMenor  $\leftarrow$  CalcularAportes(individuo)
        individuo  $\leftarrow$  individuo{elementoMenor}
      end while
    else if tamañoIndividuo<tamañoIndividuoProblema then
      elementoMayor  $\leftarrow$  CalcularMayorAporte(individuo)
      individuo  $\leftarrow$  individuo  $\cup$  {elementoMayor}
    end if
  end if
end for
```

---



---

**Algorithm 11** FuncionSolucion(individuo)

---

```
for i=0;i<numGenesIndividuo-1;i++ do
  for j=i+1;numGenesIndividuo;j++ do
    if individuo[i]==individuo[j] then
      seRepite ← true
    end if
  end for
  numGenes ← numGenes + 1
end for
if numGenes!=numGenesIndividuo then
  malTamaño ← true
end if
return !(malTamaño ∨ seRepite)
```

---

---

**Algorithm 12** CalcularAportes(individuo)

---

```
aporte ← 0
listaAportes ← ∅
for gen1 ∈ individuo do
  for gen2 ∈ individuo do
    aporte ← aporte + matrizDistancias[gen1][gen2]
  end for
  AñadirAporte(gen1,aporte)
end for
Sort(listaAportes)
return listaAportes[0]
```

---

---

**Algorithm 13** CalcularMayorAporte(individuo)

---

```
aporte ← 0
listaAportes ← ∅
for gen1 ∈ matrizDatos do
  for gen2 ∈ matrizDatos do
    aporte ← aporte + matrizDistancias[gen1][gen2]
  end for
  AñadirAporte(gen1,aporte)
end for
Sort(listaAportes)
return listaAportes[numGenesIndividuos-1]
```

---

---

**Algorithm 14** Mutar(*poblacionHijos*,*semilla*)

---

**for** *individuo*  $\in$  *poblacionHijos* **do**  
    *posMuta*  $\leftarrow$  *GeneraEnteroAleatorio*(*semilla*)  
    *eleMutado*  $\leftarrow$  *GeneraEnteroAleatorio*(*semilla*)  
    Intercambia(*individuo*,*posMuta*,*eleMutado*)  
**end for**

---

---

**Algorithm 15** ReemplazarElite(*poblacionHijos*,*elites*)

---

Sort(*poblacionHijos*)  
*indice*  $\leftarrow$  0  
**for** *elite*  $\in$  *elites* **do**  
    *poblacionHijos*[*indice*]  $\leftarrow$  *elite*  
    *indice*  $\leftarrow$  *indice* + 1  
**end for**  
**return** *poblacionHijos*

---

## 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: 80WK Lenovo Y520-15IKBN
- S.O: KDE neon User Edition 5.20 x86 64
- Kernel: 5.4.0-52-generic
- CPU: Intel i5-7300HQ (4) @ 3.500GHz
- GPU: NVIDIA GeForce GTX 1050 Mobile
- GPU: Intel HD Graphics 630
- Memoria RAM : 7837 MiB.

#### 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.

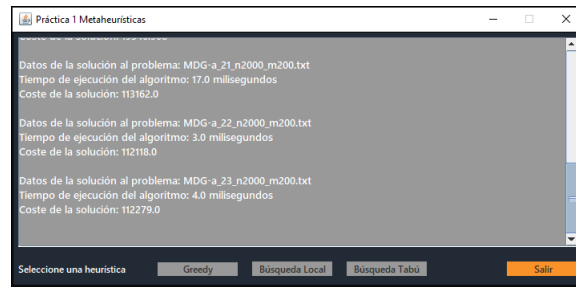


Figure 1: GUI

## 4.2 Parámetros de los algoritmos

### 4.2.1 Genetico

### 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] Fred Glover, Manuel Laguna, Rafael Martí. Principles of Tabu Search.  
<https://www.uv.es/~rmarti/paper/docs/ts1.pdf>
- [2] <https://sci2s.ugr.es/graduateCourses/Metaheurísticas>
- [3] <https://sci2s.ugr.es/graduateCourses/Algoritmica>