

---

# Práctica 1

Metaheurísticas

Amin Kasrou Aouam



**UNIVERSIDAD  
DE GRANADA**

2021-04-19

## Índice

<b>Práctica 1</b>	<b>3</b>
Introducción . . . . .	3
Algoritmos . . . . .	3
Greedy . . . . .	3
Búsqueda local . . . . .	3

## Práctica 1

### Introducción

En esta práctica, usaremos distintos algoritmos de búsqueda para resolver el problema de la máxima diversidad (MDP). Implementaremos:

- Algoritmo *Greedy*
- Algoritmo de búsqueda local

### Algoritmos

#### Greedy

El algoritmo *greedy* añade de forma iterativa un punto, hasta conseguir una solución de tamaño  $m$ .

En primer lugar, seleccionamos el elemento más lejano de los demás (centroide), y lo añadimos en nuestro conjunto de elementos seleccionados. A éste, añadiremos en cada paso el elemento correspondiente según la medida del *MaxMin*. Ilustramos el algoritmo a continuación:

---

---

**Input:** A list  $[a_i], i = 1, 2, \dots, m$ , that contains the chosen point and the distance

**Output:** Processed list

```
1  $Sel = []$ 
2  $centroid \leftarrow getFurthestElement()$ 
3 for  $i \leftarrow 0$  to  $m$  do
4   for  $element$  in  $Sel$  do
5      $closestElements = []$ 
6      $closestPoint \leftarrow getClosestPoint(element)$ 
7      $closestElements.append(closestPoint)$ 
8   end for
9    $maximum \leftarrow max(closestElements)$ 
10   $Sel.append(maximum)$ 
11 end for
12 return  $Sel$ 
```

---

#### Búsqueda local

El algoritmo de búsqueda local selecciona una solución aleatoria, de tamaño  $m$ , y explora durante un número máximo de iteraciones soluciones vecinas.

Para mejorar la eficiencia del algoritmo, usamos la heurística del primer mejor (selección de la primera solución vecina que mejora la actual). Ilustramos el algoritmo a continuación:

---

---

**Input:** A list  $[a_i], i = 1, 2, \dots, m$ , the solution  
**Output:** Processed list

```
1 Solutions = []
2 firstSolution ← getRandomSolution()
3 Solutions.append(firstSolution)
4 lastSolution ← getLastElement(neighbour)
5 maxIterations ← 1000
6 for i ← 0 to maxIterations do
7   while neighbour ≤ lastSolution do
8     neighbour ← getNeighbouringSolution(lastSolution)
9     Solutions.append(neighbour)
10    lastSolution ← getLastElement(neighbour)
11  end while
12  finalSolution ← getLastElement(Solutions)
13 end for
14 return finalSolution
```

---