



# Algoritmos Greedy (o voraces)

## Algorítmica. Práctica 3

---

Jose Alberto Hoces Castro

Javier Gómez López

Manuel Moya Martín Castaño

Mayo 2022

# Contenidos

1. Ejercicio 1. Contenedores

2. Ejercicio 2. El problema del viajante de comercio

3. Conclusiones

## Objetivo de la práctica

Aprender a analizar un problema y resolverlo mediante la técnica Greedy, además de justificar su utilidad para resolver problemas de forma muy eficiente, obteniendo la solución óptima o muy cercana a la óptima.

## **Ejercicio 1. Contenedores**

---

# Enunciado

*Se tiene un buque mercante cuya capacidad de carga es de  $K$  toneladas y un conjunto de contenedores  $c_1, \dots, c_n$  cuyos pesos respectivos son  $p_1, \dots, p_n$  (expresados también en toneladas). Teniendo en cuenta que la capacidad del buque es menor que la suma total de los pesos de los contenedores:*

## Primer ejercicio

*Diseñe un algoritmo que maximice el número de contenedores cargados, y demuestre su optimalidad.*

## Primer ejercicio. Planteamiento del algoritmo

- Como queremos cargar el máximo número de contenedores, empezaremos cargando los más **pequeños**.
- Ordenamos de **menor a mayor** peso los contenedores.
- Empezamos a cargar los de menor peso hasta que superemos las K toneladas del buque mercante.
- Todo esto lo simulamos con un vector de enteros en nuestro código, el cual tenemos a continuación.

## Primer ejercicio. Código

```
1 int contenedoresGreedy1(int *T, int n){
2
3     int used = 0;
4     int result = 0;
5     vector<int> myvector(T,T+n);
6     sort(myvector.begin(),myvector.end());
7
8     for(int i = 0; (i < n) && (used <= n); i++){
9         used += T[i];
10        result++;
11    }
12
13    return result;
14 }
```



## Primer ejercicio. Enfoque Greedy

Las 6 características de nuestro problema que hacen que lo identifiquemos como problema Greedy son:

- **Un conjunto de candidatos:** En este caso, los contenedores a cargar.
- **Una lista de candidatos ya usados:** Los contenedores que ya han sido cargados.
- **Un criterio que dice cuándo un conjunto de candidatos forma una solución:** El criterio es que la suma de los pesos de un conjunto de contenedores no sea superior a las K toneladas del buque.

## Primer ejercicio. Enfoque Greedy

- **Un criterio que dice cuándo un conjunto de candidatos es factible (podrá llegar a ser una solución):** el conjunto de contenedores que se evalúe no debe superar en peso las K toneladas del buque.
- **Una función de selección que indica en cualquier instante cuál es el candidato más prometedor de los no usados todavía:** El contenedor de menor peso de los que aún no están cargados, de ahí que los ordenemos de menor a mayor peso.
- **La función objetivo que intentamos optimizar:** El número de contenedores a cargar, es lo que queremos maximizar.

## Primer ejercicio. Estudio de la optimalidad

Sea  $T = \{c_1, \dots, c_n\}$  y llamemos  $S = \{c_1, \dots, c_m\}$  a la solución de nuestro algoritmo Greedy.

$$\sum_{c_i \in S} p_i = \sum_{i=1}^m p_i \leq K \quad \text{y} \quad \sum_{i=1}^{m+1} p_i > K$$

Sea  $U \subset T$  con un número mayor de contenedores que  $S$  y veamos que no es solución, es decir, que  $\sum_{c_i \in U} p_i > K$ .

## Primer ejercicio. Estudio de la optimalidad

$$\sum_{c_i \in U} p_i = \sum_{c_i \in S \cap U} p_i + \sum_{c_i \in U \setminus S} p_i$$

↓

$$\sum_{c_i \in S \cap U} p_i + \sum_{c_i \in U \setminus S} p_i = \sum_{c_i \in S \cap U} p_i + \sum_{c_i \in R} p_i + \sum_{c_i \in U \setminus S \setminus R} p_i$$

↓

$$\sum_{c_i \in S \cap U} p_i + \sum_{c_i \in R} p_i + \sum_{c_i \in U \setminus S \setminus R} p_i \geq \sum_{c_i \in S \cap U} p_i + \sum_{c_i \in S \setminus U} p_i + \sum_{c_i \in U \setminus S \setminus R} p_i$$

↓

$$\sum_{c_i \in S \cap U} p_i + \sum_{c_i \in S \setminus U} p_i + \sum_{c_i \in U \setminus S \setminus R} p_i = \sum_{c_i \in S} p_i + \sum_{c_i \in U \setminus S \setminus R} p_i > K$$

Luego hemos demostrado que  $\sum_{c_i \in U} p_i > K$  y por lo tanto,  $U$  no es solución.

## Segundo ejercicio

*Diseñe un algoritmo que intente maximizar el número de toneladas cargadas.*

## Segundo ejercicio. Planteamiento del algoritmo

- Como queremos cargar el máximo número de toneladas, empezaremos cargando los más **pesados**.
- Ordenamos de **mayor a menor** peso los contenedores.
- Empezamos a cargar los de mayor peso hasta que superemos las K toneladas del buque mercante.
- Todo esto lo simulamos con un vector de enteros en nuestro código, el cual tenemos a continuación.

## Segundo ejercicio. Código

```
1 int contenedoresGreedy2(int *T, int n){
2
3     int used = 0;
4     vector<int> myvector(T,T+n);
5     sort(myvector.begin(),myvector.end(), greater<int>());
6
7     for(int i = 0; (i < n) && (used <= n); i++){
8         used += T[i];
9     }
10
11     return used;
12 }
```

## Segundo ejercicio. Estudio de la optimalidad

[5, 4, 6, 1, 1, 2, 7, 9, 8, 3] K = 10



[9, 8, 7, 6, 5, 4, 3, 2, 1, 1]

**Solución aportada por nuestro algoritmo:** [9]

**Solución óptima:** [1,2,3,4]



## **Ejercicio 2. El problema del viajante de comercio**

---

# Enunciado

*Dado un conjunto de ciudades y una matriz con las distancias entre todas ellas, un viajante debe recorrer todas las ciudades exactamente una vez, regresando al punto de partida de forma tal que la distancia recorrida sea mínima.*

## TSP. Heurística del vecino más cercano

1. Partimos de un nodo cualquiera.
2. Encontramos el nodo más cercano a este nodo, y lo añadimos al recorrido.
3. Repetimos el proceso hasta cubrir todos los nodos.

# Heurística del vecino más cercano. Código

```
1 def get_best_solution(points):
2     road = []
3     order = []
4
5     distance_matrix = gen_distance_matrix(points, len(points))
6
7     #We start always at first point
8     last_point = 0
9     road.append(points[last_point])
10    order.append(0)
11
12    while len(road) < len(points):
13        best_position = get_min_row_element(distance_matrix,
14                                            last_point)
15
16        road.append(points[best_position])
17        order.append(best_position)
18
19        clean_position(distance_matrix, last_point)
20
21        last_point = best_position
22
23    road_distance = get_road_distance(road)
24
25    return road, road_distance, order
```

# Heurística del vecino más cercano. Análisis Teórico

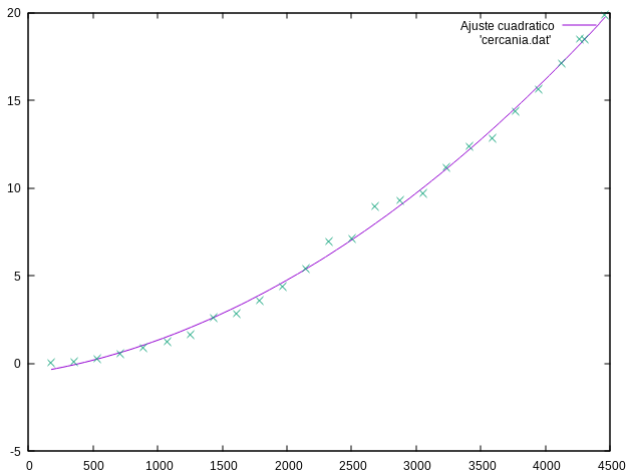
$$T(n) \in O(n^2)$$

# Heurística del vecino más cercano. Análisis empírico

Heurística del Vecino más cercano	
Ciudades (n)	Tiempo (s)
358	0.134928381001373
537	0.301542887000323
716	0.571918544999789
895	0.886006714001269
1074	1.24296611099999
1253	1.64954555899931
1432	2.64052301800075
1611	2.85592838900084
1790	3.59707787300067
1969	4.38948754500052
2148	5.39098084799844
2327	6.98254896399885
2506	7.10732670200014
2685	8.94702127299934
2874	9.27537115900122
3053	9.69826381100029
3232	11.1923151139999
3411	12.4046790310003
3590	12.853421451
3769	14.3906136509995
3948	15.625681644
4127	17.1084850459993
4306	18.5073586279996
4461	19.8675596370013

**Tabla 1:** Experiencia empírica de el vecino más cercano

# Heurística del vecino más cercano. Análisis Híbrido



**Figura 1:** Gráfica con los tiempos de ejecución del vecino más cercano.

$R^2 = 0.869828$

## Heurística del vecino más cercano. Resultados

- ulysses16.tsp:

[0, 7, 15, 12, 11, 13, 6, 5, 14, 4, 8, 9, 3, 1, 2, 10]       $D = 103$

- bayg29.tsp:

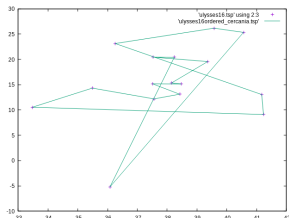
[0, 27, 5, 11, 8, 4, 20, 1, 19, 9, 3, 14, 17, 13, 21, 16, 10, 18, 24,  
6, 22, 26, 7, 23, 15, 12, 28, 25, 2]       $D = 10209$

- eil76.tsp:

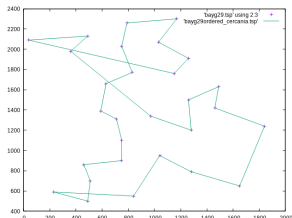
[0, 72, 32, 62, 15, 2, 43, 31, 8, 38, 71, 57, 9, 37, 64, 10, 65, 52, 13, 18, 34, 6, 7, 45,  
33, 51, 26, 44, 28, 47, 46, 20, 73, 27, 61, 1, 29, 3, 74, 75,  
66, 25, 11, 39, 15, 50, 5, 67, 4, 36, 19, 69, 59, 70, 35, 68, 60, 2, 1,  
41, 40, 42, 22, 55, 48, 23, 17, 49, 24, 54, 30, 58, 53, 12, 56, 14, 63]       $D = 642$



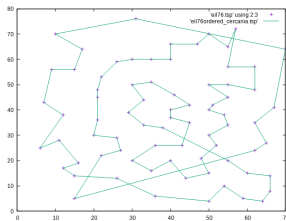
# Heurística del vecino más cercano. Gráficos



Ulysses



Bayg



Eil

# TSP. Heurística de inserción

Empezar insercion aqui

## TSP. Heurística de perturbaciones

Este enfoque, de nuevo *greedy*, realiza las perturbaciones indicadas por un parámetro sobre un recorrido dado para intentar mejorarlo.

# Heurística de perturbaciones. Código

```
1 def perturbate(road, orden, pos):
2     current_perb = road
3     best_gain = math.inf
4     best_perturbation = pos
5     base_distance = get_road_distance(road)
6
7     for i in range(len(road)):
8         current_perb = get_swap(road, pos, i)
9         swap_distance = get_road_distance(current_perb)
10
11         if swap_distance < base_distance:
12             best_perturbation = i
13             base_distance = swap_distance
14
15     road = get_swap(road, pos, best_perturbation)
16     orden = get_swap(orden, pos, best_perturbation)
17
18 def get_best_solution_perturbations(points, orden, perturbations
19 ):
20     base_road = points
21
22     for i in range(perturbations):
23         pos = get_worst_node(points)
24
25         perturbate(base_road, orden, pos)
26
27     return base_road, get_road_distance(base_road), orden
```

# Heurística de perturbaciones. Análisis teórico

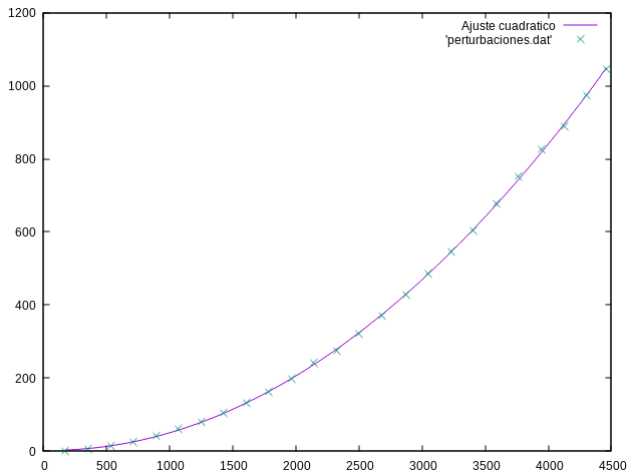
$$T(n) \in O(n^2 \cdot \text{perturbaciones}) \Rightarrow T(n) \in O(n^2)$$

# Heurística de perturbaciones. Análisis empírico

Heurística de perturbaciones	
Ciudades (n)	Tiempo (s)
358	7.09205518299859
537	14.8038363970009
716	27.0365697790003
895	42.5021277929991
1074	60.6207957549996
1253	79.9574952149997
1432	106.705971191001
1611	133.111995577001
1790	163.618691645999
1969	198.794982856998
2148	241.148463359001
2327	275.541576210999
2506	323.217905321999
2685	371.290206549998
2874	429.912281959001
3053	485.322751331001
3232	545.820733338998
3411	603.358232573999
3590	678.706273265001
3769	751.554062298001
3948	825.585648235003
4127	890.451960293001
4306	973.566447267
4461	1045.84781561849

**Tabla 2:** Experiencia empírica de perturbaciones

# Heurística de perturbaciones. Análisis híbrido



**Figura 2:** Gráfica con los tiempos de ejecución del vecino más cercano.

$R^2 = 0.93507$

## Heurística de perturbaciones. Resultados

- ulysses16.tsp:

[0, 7, 15, 12, 11, 13, 6, 5, 14, 4, 8, 9, 2, 1, 3, 10]       $D = 101$

- bayg29.tsp:

[0, 27, 5, 11, 8, 4, 20, 1, 19, 9, 3, 14, 17, 13, 21, 16, 10, 18, 24, 6, 22, 26, 7,

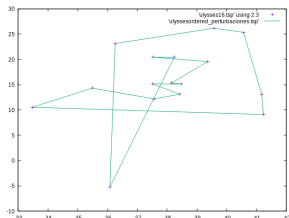
23, 15, 12, 28, 25, 2]       $D = 10209$

- eil76.tsp: Aplicando 10 perturbaciones, obtenemos que el mejor orden (teniendo en cuenta el orden del fichero original):

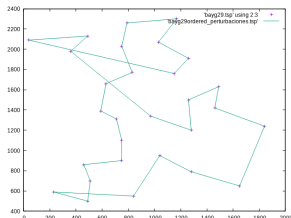
[0, 72, 32, 62, 15, 2, 43, 31, 8, 38, 71, 57, 9, 37, 64, 10, 65, 52, 13, 18, 34, 6, 7, 45, 33, 51, 26, 44, 28,  
47, 46, 20, 73, 27, 61, 1, 29, 3, 74, 75, 66, 25, 11, 39, 16, 50, 5, 67, 4, 36, 19, 69, 59, 70, 35, 68, 60, 2,  
1, 41, 40, 42, 22, 55, 48, 23, 17, 49, 24, 54, 30, 58, 53, 12, 56, 14, 63]       $D = 642$



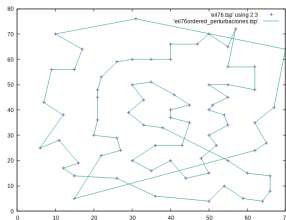
# Heurística de perturbaciones. Gráficos



Ulysses



Bayg



Eil

# Comparación de las distintas heurísticas

a

## **Conclusiones**

---