



UNIVERSIDAD DE GRANADA

Escuela Técnica Superior de Ingeniería Informática y
Telecomunicaciones

PRÁCTICA 3: ALGORITMOS GREEDY

Doble Grado Ingeniería Informática y Matemáticas

Autores:

Jose Alberto Hoces Castro

Javier Gómez López

Moya Martín Castaño

Mayo 2022



Este trabajo se distribuye bajo una licencia CC BY-NC-SA 4.0.

Eres libre de distribuir y adaptar el material siempre que reconozcas a los autores originales del documento, no lo utilices para fines comerciales y lo distribuyas bajo la misma licencia.

creativecommons.org/licenses/by-nc-sa/4.0/

Índice

1. Introducción	3
2. Desarrollo	3
2.1. Ejercicio 1. Contenedores	3
2.1.1. Primer ejercicio	3
2.1.2. Segundo ejercicio	4
2.2. Ejercicio 2. El problema del viajante de comercio	5

1. Introducción

El objetivo de esta práctica es aprender a implementar y utilizar algoritmos “*greedy*” o voraces para resolver problemas de manera rápida aunque no por ello menos óptima. Para ello, se plantean los siguientes dos problemas:

- **Ejercicio 1** (Contenedores): Se quiere rellenar un buque mercante con una cierta capacidad de peso con contenedores, cada uno de los cuales tiene su propio peso.
- **Ejercicio 2** (TSP): El problema del viajero. Se quiere recorrer una serie de ciudades, pasando por ellas solo una vez y volviendo al punto de partida. Se quiere encontrar la ruta más óptima.

2. Desarrollo

Para el primer ejercicio, nos centraremos en identificar el problema de los contenedores como un problema Greedy identificando sus características. También, justificaremos la optimalidad o la no optimalidad de los algoritmos Greedy desarrollados.

Para el análisis de los algoritmos del viajante de comercio que desarrollaremos, hemos realizado los siguientes pasos:

1. Un **análisis teórico** de los algoritmos usando las técnicas vistas en clase.
2. Un **análisis empírico** donde hemos ejecutado los algoritmos en nuestros ordenadores bajo las mismas normas y condiciones. Hemos compilado usando la optimización `-Og`. Además, hemos usado como *datasets* de pruebas los datos proporcionados por la profesora en el caso del TSP, y valores aleatorios de los pesos de los contenedores para el primer problema. Por otro lado, para automatizar el proceso, hemos creado unos *scripts* de generación de datos de prueba y de ejecución de nuestros programas. Hemos ejecutado cada algoritmo 15 veces en cada uno de los tamaños probados, y hemos hecho la media de ellos para reducir perturbaciones que puedan alterar el resultado.
3. Un **análisis híbrido** donde hemos tomado los datos de cada uno de los alumnos del grupo y hemos hallado la K (constante oculta). Para ello hemos usado `gnuplot`.

2.1. Ejercicio 1. Contenedores

El enunciado del problema es el siguiente: *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:*

- Diseñe un algoritmo que maximice el número de contenedores cargados, y demuestre su optimalidad.
- Diseñe un algoritmo que intente maximizar el número de toneladas cargadas.

2.1.1. Primer ejercicio

Nuestro objetivo es que podamos cargar el máximo número de contenedores en un buque mercante de K toneladas. Para ello, el algoritmo que nosotros proponemos es tomar los contenedores, ordenarlos de menor a mayor peso, y comenzar añadiendo los de menor peso. De esta forma, podemos cargar más contenedores, ya que si empezásemos por los de pesos intermedios o mayores, acabaríamos cargando menos.

Veamos las 6 características de nuestro problema Greedy:

- **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.
- **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 prome-**
tedor 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.

A continuación pasamos a mostrar el código de nuestro algoritmo. Para representar el conjunto de contene-
dores, hemos considerado un vector de enteros, siendo cada entero el peso de cada contenedor. Para asegurarnos
de que la suma de los pesos de todos los contenedores supera a las K toneladas del buque, el vector es de
dimensión K con enteros aleatorios desde 0 hasta K (en nuestro código lo representamos por el parámetro n,
introducido por el usuario). Nuestra función devuelve el número de contenedores que se han podido cargar:

```
int contenedoresGreedy1(int *T, int n){
    int used = 0;
    int result = 0;
    vector<int> myvector(T,T+n);
    sort(myvector.begin(),myvector.end());

    for(int i = 0; (i < n) && (used <= n); i++){
        used += T[i];
        result++;
    }

    return result;
}
```

- **Estudio de la optimalidad**

2.1.2. Segundo ejercicio

En este segundo ejercicio, lo que queremos es maximizar las toneladas cargadas en el buque sin sobrepasar
su capacidad total. Para ello, seguimos el pensamiento inverso al planteado en el anterior ejercicio. Como lo que
nos interesa es cargar el máximo de toneladas posibles, empezaremos cargando aquellos contenedores cuyo peso
sea el más grande. Para ello, los ordenamos de mayor a menor peso, justo al contrario que antes. Identificamos
las 6 características de un problema Greedy:

- **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.
- **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 prome-**
tedor de los no usados todavía: El contenedor de mayor peso de los que aún no están cargados, de
ahí que los ordenemos de mayor a menor peso.
- **La función objetivo que intentamos optimizar:** El número de toneladas a cargar, es lo que queremos
maximizar.

En el código del algoritmo, al igual que antes, simulamos los contenedores con sus respectivos pesos con un
vector de enteros, el cual ordenamos de mayor a menor usando el sort de la STL. Se van sumando los pesos
hasta que se sobrepase el tope de toneladas del buque. Nuestra función devuelve el número de toneladas que se
han podido cargar en total:

```
int contenedoresGreedy2(int *T, int n){
    int used = 0;
    vector<int> myvector(T,T+n);
    sort(myvector.begin(),myvector.end(), greater<int>());

    for(int i = 0; (i < n) && (used <= n); i++){
```

```
        used += T[i];  
    }  
    return used;  
}
```

■ Estudio de la optimalidad

Sin embargo, en este caso nuestro algoritmo no nos da la solución óptima. Veámoslo con un contraejemplo. Con $n = 10$, imaginemos que tenemos el vector $[5, 4, 6, 1, 1, 2, 7, 9, 8, 3]$. Nuestro algoritmo lo ordenaría de mayor a menor, obteniendo el vector $[9, 8, 7, 6, 5, 4, 3, 2, 1, 1]$. Tras esto, se incluiría el contenedor de peso 9, pero el siguiente ya no sería posible cargarlo ya que $9 + 8 = 17 > 10$. La solución óptima en este caso sería tomar un contenedor de peso 1, otro de 2, otro de 3 y otro de 4, aprovechando así las 10 toneladas en su totalidad.

2.2. Ejercicio 2. El problema del viajante de comercio