UNIVERSIDAD DE GRANADA

E.T.S. DE INGENIERÍAS INFORMATICA y DE TELECOMUNICACIÓN

Departamento de Ciencias de la Computación e Inteligencia Artificial

Práctica 3: Algoritmos Voraces (Greedy)

grupo b4

Alejandro Casado Quijada

Andrés Ortiz Corrales

Antonio Jiménez Martínez

Jesús Prieto López

Salvador Rueda Molina

Curso 2013-2014

Grado en Informática

1. Objetivo

El objetivo de esta práctica es que el estudiante aprecie la utilidad de los métodos voraces (greedy) para resolver problemas de forma muy eficiente, en algunos casos obteniendo soluciones óptimas y en otros aproximaciones.

2. Contenedores en un barco

Se tiene un buque mercante cuya capacidad de carga es de K toneladas y un conjunto de contenedores c1 , . . . , cn cuyos pesos respectivos son p1 , . . . , pn (expresados también en toneladas).
Teniendo en cuenta que la capacidad del buque es menor que la suma total de los pesos de los contenedores:

Diseñamos un algoritmo que maximice el número de contenedores cargados, y demostramos que es óptimo.

Diseñamos un algoritmo que maximice el número de toneladas cargadas. A continuación demostraremos que no siempre encuentra el óptimo.

Mostramos el enfoque greedy:

- -conjunto de candidatos: los contenedores de entrada.
- -conjunto de seleccionados: los contenedores que se cargan en el barco.
- -Función solución:Si la suma de los pesos de los contenedores ha llegado a la capacidad máxima.
- -Función de factibilidad: comprobamos si al añadir un contenedor superamos la capacidad máxima del barco.
- -Función de selección: Es diferente en para los dos algoritmos:
  - -Maximizar el número de contenedores: Seleccionamos los que pesen menos.
  - -Maximizamos el número de toneladas: seleccionamos los que pesen más.
- -Función objetivo:Es diferente en para los dos algoritmos:
  - -Maximizar el número de contenedores
  - -Maximizamos el número de toneladas

También creamos un a función para generar un vector aleatorio entre valores de 1-100.

```
vector<int> genera(int n) {
   vector<int> T(n);
   srand(time(0));
   for(int i=0; i<n; i++) {
      T[i]=(rand()%100)+1;//aleatorio entre 1-100
   }
   return T;
}
```

2.1. Código de los algoritmos.

-Maximizar el número de contenedores

```
int seleccion(const vector <int> &c) {
   int min=c[0];
   int minpos=0;
   int p;
   for(int i=1; i<c.size(); i++) {
      p=c[i];
      if(min >=p) {
         min=p;
         minpos=i;
      }
   }
   return minpos;
}
//entrada vector de contenedores y el tamaño maximo
//devuelve un vector con los contenedores
vector <int> Barco(vector <int> c ,int k) {
   vector<int> s;
   int obj;
   int x;//posicion del vector de candidatos
   int peso_actual=0;
   while(!c.empty() && peso_actual<=k) {
      x= seleccion(c);
      obj =c[x];
      c[x]=c[c.size()-1];
      c.pop_back();
      if((peso_actual + obj) <= k) { //funcion de factibilidad
         s.push_back(obj);
         peso_actual+=obj;
      }
   }
   return s;
}
```

-Maximizamos el número de toneladas

```
int seleccion(const vector <int> &c) {
    int max=c[0];
    int maxpos=0;
    int p;
    for(int i=1; i<c.size(); i++) {
        p=c[i];
        if(max <=p) {
            max=p;
            maxpos=i;
        }
    }
    return maxpos;
}
//entrada vector de contenedores y el tamaño máximo
//devuelve un vector con los contenedores
vector <int> Barco(vector <int> c ,int k) {
    vector<int> s;
    int obj;
    int x;//posicion del vector de candidatos
    int peso_actual=0;
    while(!c.empty() && peso_actual<=k) {
        x= seleccion(c);
        obj =c[x];
        c[x]=c[c.size()-1];
        c.pop_back();
        if((peso_actual + obj) <= k) { //funcion de factibilidad
            s.push_back(obj);
            peso_actual+=obj;
        }
    }
    return s;
}
```

2.2. Demostración de los algoritmos.

-Maximizamos el número de toneladas:

Ordenamos el vector de forma decreciente. Y vamos cogiendo los elementos mayores. De manera que vamos tomando el máximo tamaño.
Ponemos un ejemplo de un comportamiento óptimo.
Peso máximo=100.
Contenedores: 50,30,15,10.
Los contenedores seleccionados son: 50,30,15
como podemos ver encuentra el óptimo. Pero no siempre ya que podemos encontrar un contraejemplo que demuestra que el algoritmo greedy no siempre es eficaz.
Peso máximo=100.
Contenedores: 40,30,20,7,5,4
Los contenedores seleccionados son: 40,30,20,7. Tamaño total=97.
La solución óptima sería: 40,30,20,5,4. Tamaño total=99.

De esta manera no siempre encontramos el óptimo.

-Maximizar el número de contenedores:

Ordenamos los contenedores de forma creciente. a<=b<=c<=d<=Xn
Dada la situación de peso actual=P y peso máximo=Pmax.
P es la sumatorio de los pesos seleccionados.
Dadas 4 situaciones:

a)p+c>Pmax → p+d>Pmax. P sería el peso solución.

b)P+c<=Pmax
P+d>Pmax → P+Xn>Pmax.
P+c es la solución.

c)P+c<=Pmax
P+d<=Pmax.
P+c+d<=Pmax. Sumaríamos c y d a P.

d)P+c<=Pmax
P+d<=Pmax.
P+c+d>Pmax → p+c+Xn>Pmax.
Suponemos un α para el cual p+c+α<=Pmax → α<d→ α<Xn
α<=c sin embargo si existe α ya esta incluido en P.

Esto muestra que si se cumple para c, se cumple para de y asi para todos:
d>=c
p+c+Xn>Pmax
p+d+Xn>Pmax.

De esta manera demostramos que este algoritmo siempre encuentra el máximo.

3. Cálculo del tiempo teórico

Tenemos dos algoritmo para este problema. Ambos de orden cuadrático
-Maximizar el número de contenedores
-Maximizamos el número de toneladas

Ya que en cada iteración recorre el vector de candidatos.

4. Cálculo del tiempo Empírica

Realizaremos el cálculo de la eficiencia empírica estudiando el comportamiento de los dos algoritmos: maximizar el número de contenedores y maximizamos el número de toneladas

Con el uso del archivo de entradas proporcionado mediremos los tiempos de ejecución para cada tamaño de entrada. Será mediante entradas de vectores aleatorios.

Para el cálculo empírico nos hemos ayudado de la biblioteca STL::chrono para una mayor precisión en el tiempo.

Hemos creado un script para ejecutar los distintos algoritmos con diferentes tamaños de entradas.

5. Cálculo del tiempo Híbrida

Para conocer la forma mas exacta de la ecuación del tiempo de ejecución para una situación concreta se utiliza el calculo de la eficiencia híbrida. A la hora de realizar el cálculo teórico se presenta cada término con una constante de valor desconocido.

Entonces necesitamos saber el valor de esas constantes, ajustando la función a un conjunto de puntos. Se utiliza la función del cálculo teórico y el conjunto de puntos lo forman los resultados del análisis empírico y se ajusta con regresión de mínimos cuadrados.

Hemos utilizado las diferentes funciones de regresión para las diferentes gráficas:

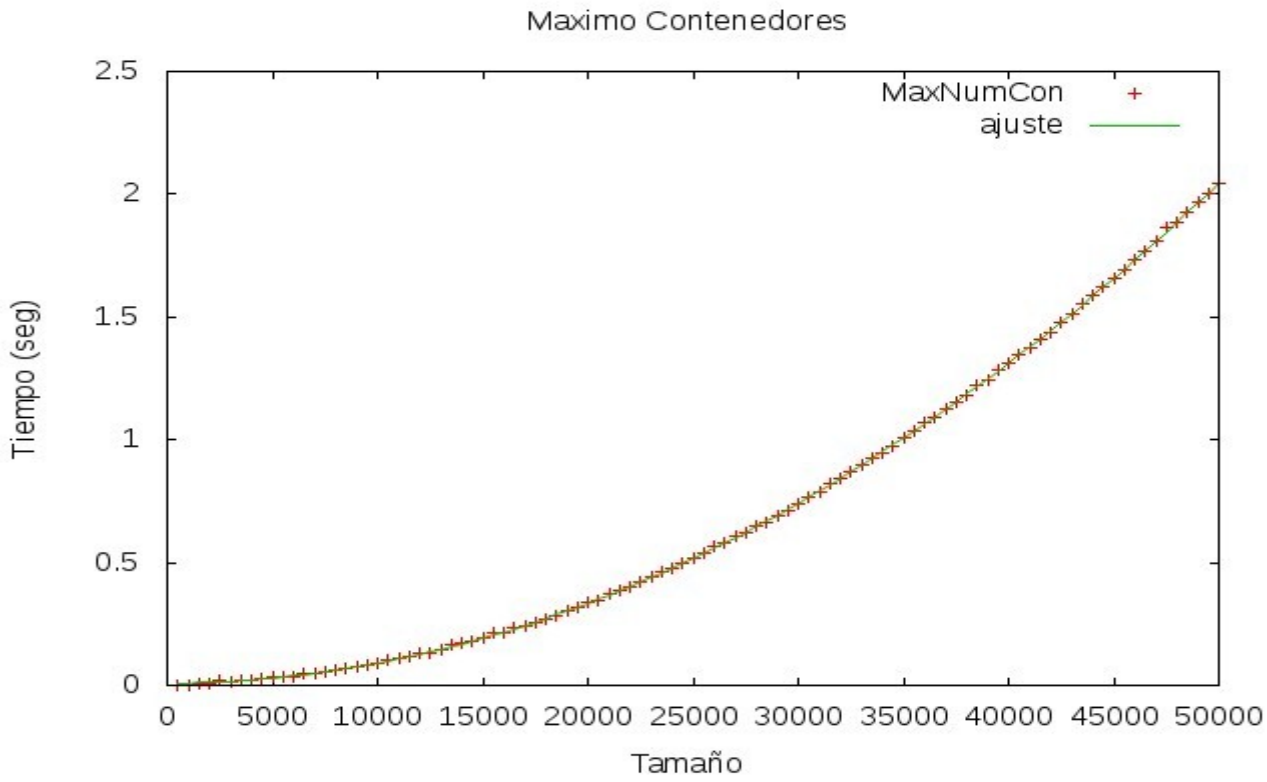
-Cuadrático: a0*x*x+a1*x+a2

Cuando decimos que aplicamos el algoritmo a <<una situación concreta>> nos referimos a por ejemplo:

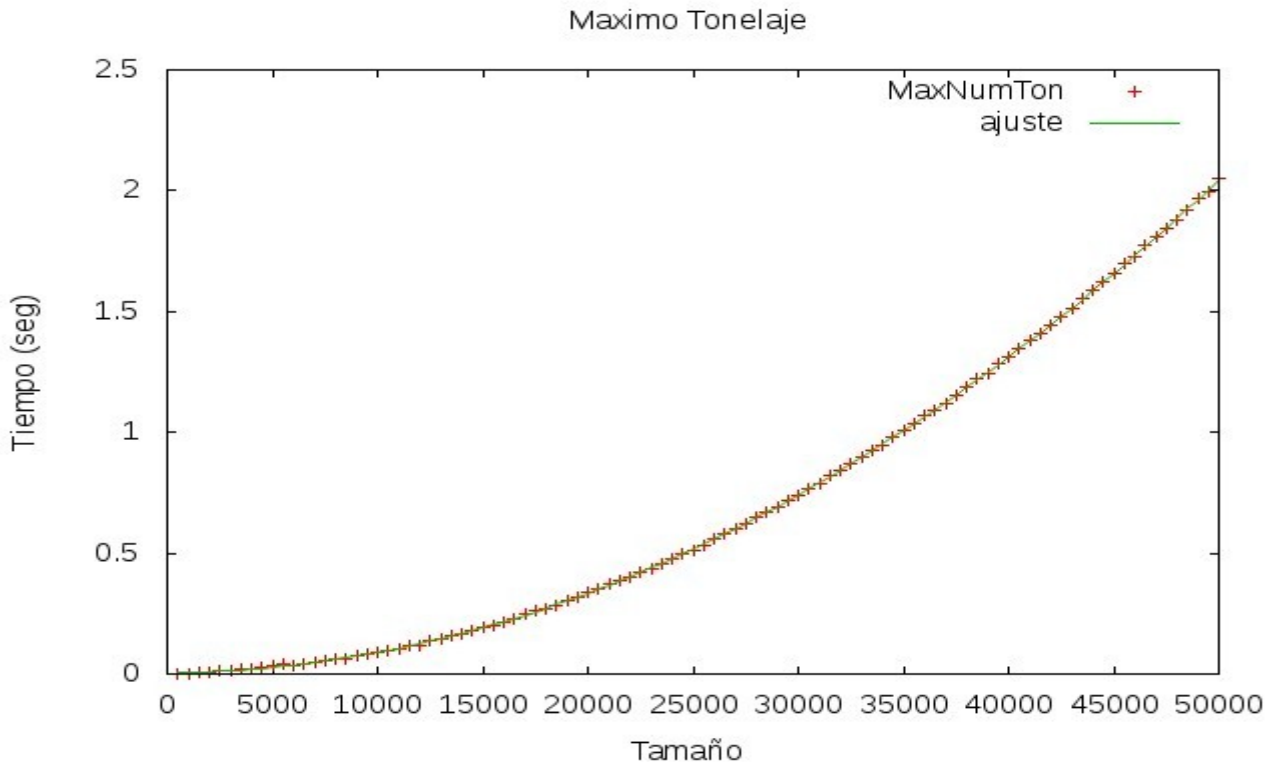
- Compilador utilizado: g++
- Ordenador sobre el que se ejecuta, en nuestro caso con la siguientes características:
  - Intel core i3 m330 2.13 Ghz
  - 4 GB RAM
  - S.O. Ubuntu 13.04 64 bits

A continuación le mostraremos las siguientes gráficas y tablas de los diferentes algoritmos. También realizaremos una comparación gráficas del algoritmo de fuerza de orden lineal y el algoritmo de divide y vencerás de orden logarítmico.

Las tablas y gráficas de los algoritmos son:
-Maximizar el número de contenedores



-Maximizamos el número de toneladas



MaxNumContenedores	
Tamaño	Tiempo
500	0.000561
1000	0.002134
1500	0.004452
2000	0.008014
2500	0.017626
3000	0.016897
3500	0.023474
4000	0.023405
4500	0.027429
5000	0.032422
5500	0.032291
6000	0.035461
6500	0.047799
7000	0.049895
7500	0.054502
8000	0.060343
8500	0.06681
9000	0.077691
9500	0.083989
10000	0.088736
10500	0.100257
11000	0.1108
11500	0.120167
12000	0.134141
12500	0.131977
13000	0.144566
13500	0.162497
14000	0.171993
14500	0.181277
15000	0.193712
15500	0.211143
16000	0.213617
16500	0.233287
17000	0.241326
17500	0.253209
18000	0.271088
18500	0.286396
19000	0.300676
19500	0.319963
20000	0.336466
20500	0.343597
21000	0.369481
21500	0.385412
22000	0.398751
22500	0.421297
23000	0.442892
23500	0.460203
24000	0.473216
24500	0.495574
25000	0.516811

MaxNumToneladas	
Tamaño	Tiempo
500	0.000552
1000	0.0021
1500	0.004417
2000	0.007945
2500	0.012093
3000	0.017171
3500	0.018772
4000	0.020892
4500	0.026583
5000	0.03195
5500	0.038805
6000	0.037945
6500	0.042419
7000	0.047017
7500	0.05393
8000	0.05927
8500	0.064882
9000	0.076379
9500	0.085872
10000	0.091306
10500	0.093289
11000	0.105964
11500	0.117085
12000	0.119834
12500	0.136361
13000	0.147564
13500	0.160624
14000	0.16883
14500	0.18094
15000	0.193479
15500	0.200259
16000	0.216223
16500	0.226417
17000	0.245652
17500	0.261753
18000	0.26814
18500	0.28426
19000	0.300424
19500	0.31928
20000	0.341763
20500	0.353802
21000	0.374283
21500	0.386798
22000	0.400492
22500	0.421098
23000	0.43698
23500	0.457284
24000	0.479471
24500	0.497312
25000	0.509621

25500	0.541221
26000	0.566213
26500	0.578624
27000	0.604375
27500	0.624877
28000	0.650753
28500	0.665201
29000	0.690743
29500	0.713431
30000	0.739504
30500	0.763927
31000	0.784831
31500	0.818712
32000	0.843654
32500	0.869131
33000	0.901224
33500	0.92509
34000	0.948319
34500	0.970347
35000	1.008
35500	1.0344
36000	1.06987
36500	1.09261
37000	1.12852
37500	1.15196
38000	1.18281
38500	1.21984
39000	1.24307
39500	1.28498
40000	1.31181
40500	1.34831
41000	1.37353
41500	1.40939
42000	1.43801
42500	1.47919
43000	1.51479
43500	1.55472
44000	1.58531
44500	1.62055
45000	1.6563
45500	1.69484
46000	1.73462
46500	1.77137
47000	1.80879
47500	1.8617
48000	1.88392
48500	1.92363
49000	1.96954
49500	2.00056
50000	2.04409

25500	0.535033
26000	0.557775
26500	0.581692
27000	0.598648
27500	0.624102
28000	0.650107
28500	0.668239
29000	0.691571
29500	0.714979
30000	0.740145
30500	0.763995
31000	0.790117
31500	0.818898
32000	0.842477
32500	0.869645
33000	0.898146
33500	0.924393
34000	0.947716
34500	0.980605
35000	1.0089
35500	1.03275
36000	1.06728
36500	1.09064
37000	1.11989
37500	1.15292
38000	1.18779
38500	1.22567
39000	1.24505
39500	1.28161
40000	1.31357
40500	1.34563
41000	1.3827
41500	1.41125
42000	1.44482
42500	1.47943
43000	1.51004
43500	1.55521
44000	1.58498
44500	1.62186
45000	1.65714
45500	1.69752
46000	1.72802
46500	1.7734
47000	1.80978
47500	1.84392
48000	1.88121
48500	1.92018
49000	1.96648
49500	1.99882
50000	2.04899