Assignment n1

Andrea Rubbi

May 2019

1 The problem

A biologist will soon move to the franco-italian Antarctic Station1. The Station is not covered by broadband Internet, and so most of the needed data must be brought with him. He can bring with him only one hard disk with a capacity of n gigabytes. Inside the hard disk he need to put as many of his databases as possible. He's currently working with m different databases, call them 1;... ;m, with database j having a size equal to Sj megabytes. The total amount of gigabytes is much bigger than n gigabytes How should one decide which ones of the databases he should bring with him? One first attribute to each database a measure Uj of the utility the database has for him. Then, he realizes he wants to bring a set of databases with maximal total quality among those which fits into the hard disk.

2 Mathematical formulation

Input: a list of m elements, where m is an integer. Each jth element has a value Uj and a size Sj. U and S are lists of integers. The total capacity is equal to n, which is an integer value.
Output: E, a subset of 1;....;m elements such that: $\sum_{\mathrm{i}\epsilon \mathrm{E}} S_{\mathrm{i}} <= n$ and $\sum_{\mathrm{i}\epsilon \mathrm{E}} U_{\mathrm{i}}$ is maximal

3 Exhaustive search algorithm

3.1 Pseudocode

```
FilltheSpace(n,Ls,Lu,E,val):
    i ← |Ls|
    if p=0 or n=0:
        return E,val
    if (last element of Ls)>n:
        return FilltheSpace(n,Lu,Ls,E,val)
    else:
        return max( FilltheSpace(n-$Ls_{last}$,$Lu_{-last}$ ,$Ls_{-last}$,E+i,val+$Lu_{last}$)
        ,FilltheSpace(n,Lu,Ls,E,val))
```

3.1.1 LEGEND:

n=remaining space
Ls=list of the sizes of elements
Lu=list of the values of elements
E=list of elements inserted in space
val=value in space
max=takes the element with the highest val
last=last element of something
for instance Ls_{last} is the last element of Ls
-last=removed the last element of something
for instance Ls_{-last} means that from now on Ls
will be without that last element

3.2 Explanation and Complexity of the Algorithm:

3.2.1 Explanation:

The functioning of the algorithm is based on the assumption that the problem itself can be decomposed into m identical questions:'should the element be added to E?'
So every time the FilltheSpace function will be called, it will create two branches: one where the last element of Ls has been added, and one where it hasn't. Moreover, it will check whether there is enough space for that element in the remaining space or not, and, in that case, it will create a single branch where the element has not been added.
finally, when n=0 or p=0, so when the space is full or when all the elements have been analyzed, it will return the total value in space (val) and the list of elements in the space (E).
For each pair of leaves of the formed tree only the one with the highest value will

be returned, so at the end it will be outputted the best value and its correlated set of elements.

3.2.2 Complexity:

In the worst case the function will be called 2^{m+1}-1 times; hence time complexity is approximated to $O(2^m)$

4 Branch and Bound implementation

4.1 Pseudocode

```
FilltheSpace(n,Ls,Lu,E,val,Q):
    i ← |Ls|
    if p=0:
        return E,val
    if $Ls_{last}$>n:
        Q ← Q + Optimistic(n,$Ls_{-last}$,$Lu_{-last}$,E,val)
    else:
        Q ← Q + Optimistic(n,Ls,Lu,E+i,val) + Optimistic(n,$Ls_{-last}$,$Lu_{-last}$,E,val)
    best ← max(Q)
    return FilltheSpace($n_{best}$,$Ls_{best}$,$Lu_{best}$,$E_{best}$,$val_{best}$,$Q_{-best}$)
```

```
(auxiliary function)   Optimistic(n,Ls,Lu,E,val):
    i ← |Ls|
    opt ← val
    j ← n
    while p>0 and j>$Ls_i$:
        j ← j-$Ls_i$
        opt ← opt-$Lu_i$
        i ← i-1
    if i>0:
        opt ← opt+j/$Ls_i$*$Lu_i$
    return (opt,n,Ls,Lu,E,val)
```

```
(auxiliary function)   Sort(n,Ls,Lu,E,val):
1      arr ← []
2      $Ls_{new}$ ← []
3      $Lu_{new}$ ← []
4      for i←1 to |Ls|:
5          arr ← arr + ($Lu_i/Ls_i$,$Ls_i$,$Lu_i$)
6      arr ← sorted(arr)
7      for i←1 to |Ls|:
8          $Ls_{new}$ ←$Ls_{new}$+ $arr_{i;2}$
9          $Ls_{new}$ ←$Ls_{new}$+ $arr_{i;3}$
10     return FilltheSpace(n,$Ls_{new}$,$Lu_{new}$,[],0,[])
```

4.1.1 LEGEND:

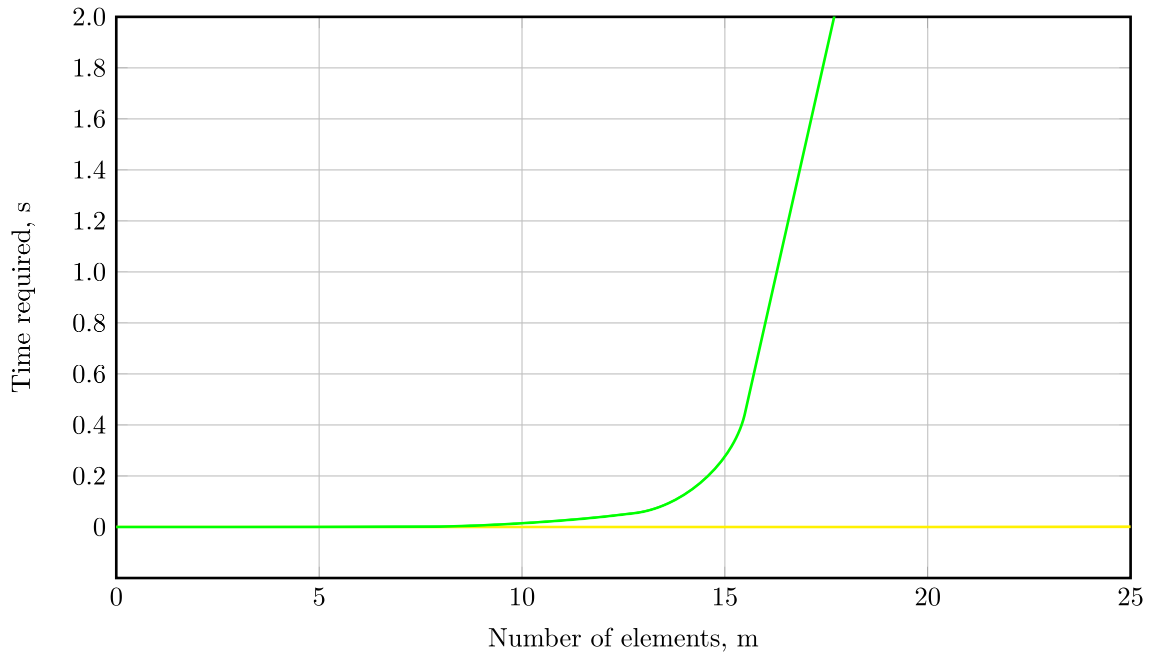
NB: read before the previous legend (of Exhaustive Search algorithm)
Q= list of optimistic values with the relative parameters (n,Ls,Lu,E,val)
max= takes the element with the highest opt
best= as subscript it indicates that the value is the one from the best element of Q. For instance Ls_{best} is the Ls taken from best
sorted(arr)= sorted arr considering the Lu/Ls ratio as parameter
Q_{-best}= Q without its best element

4.2 Explanation of the improvement:

As stated before, the problem consists on adding whole elements to the subset E. However, what if a 'fractional' adding was allowed? Then obviously the final result would be greater or equal to the one obtainable with the whole adding. So, considering the elements with the highest value/size ratio first, and adding them considering also their fractions, the algorithm can estimate an optimistic value through a branch in a fast way and then decide if that one is the most promising branch. Optimistic calculates this optimistic value, then 'opt' is returned to the main function and added to a list of 'possible candidate branches' which is then explored to find the most promising one. The 'best' candidate is the source of the values for the next call of the function. Eventually, when the length of Ls is equal to 0, 'val' and 'E' are returned.
NB: 'Sort' must be called at the begin to sort the values in the previously explained manner, then the function itself will call the main function 'FilltheSpace'.

5 Comparison

Here there is a graphical comparison between the Brute Force algorithm (Green) and the BranchBound implementation (Yellow) for n = 7m


Time required, s
Number of elements, m