

Algorithms and Data Structures for Biology

6 May 2019 — Lab Session

Ugo Dal Lago

Thomas Leventis

1 Average Case Analysis

Please refine the average case analysis of the branch-and-bound algorithm you have defined as part of the Fifth Assignment. More specifically, you are asked to consider the following situations:

- The utility is not a random number, but is somehow correlated to the size: the greater the size, the greater the utility. Suppose, in particular, that once the size has been generated randomly between 1 and 10, the utility is twice as big as the utility, plus some “noise” produced itself randomly between -3 and $+3$. Can you think of a better algorithm which can take advantage of this specific form of distribution? Please implement it.
- n is not only $3m$ or $7m$ but can be any expression between in the form im where $i \in \{1, \dots, 10\}$ For each of these expression, repeat the whole experiment. What’s the worst possible value of im ?

2 On the Fifth Assignment

Below, you can find *one possible* solution to the Fifth assignment. Of course, this is not the only possible way the assignment could be carried out.

2.1 Formal Problem

Given two positive integers m and n and two families of positive integers S_1, \dots, S_m and U_1, \dots, U_m , we want to find a set \mathcal{D} of integers between 1 and m such that $\sum_{j \in \mathcal{D}} S_j \leq n$ and $\sum_{j \in \mathcal{D}} U_j$ is maximal.

2.2 Exhaustive Search

First of all we define a method SumIndex to compute the sum of all elements in a family L whose indices are in a set D . This way given sizes S and utilities U , the total size associated to a set of databases D is SumIndex(D, S) and the total utility is SumIndex(D, U).

Algorithm 1 SumIndex(D, L)

Require: L family of integers, D set of indices of L .

Ensure: Returns the sum of all element of L with indices in D

```
s ← 0
for j in D do
  s ← s + Lj
end for
return s
```

Then to perform an exhaustive search we need to enumerate all possible sets \mathcal{D} of integers between 1 and m . Then we just need to compute their total sizes and utilities and pick a valid one

maximising utility. To do so we use an intermediate method `RestrictedExhaustiveSearch` which computes the best solution for the parameters m , n , S and U with an additional constraint: we have already decided whether we want to pick each of the first i databases, and D is the set of databases with index smaller than i which we decided to pick.

Algorithm 2 `RestrictedExhaustiveSearch(m, n, S, U, i, D)`

Require: m, n integers, S, U families of integers of size m , $i \leq m$ and D set of integers smaller than i .

Ensure: Returns a set D' of integers between 1 and m such that all $j \leq i$ is in D' if and only if it is in D , $\sum_{j \in D'} S_j \leq n$ and $\sum_{j \in D'} U_j$ is maximal if such a set exists, and returns the emptyset otherwise.

```

if  $i = m$  then
  if  $\text{SumIndex}(D, S) \leq n$  then
    return  $D$ 
  else
    return  $\emptyset$ 
  end if
else
   $D1 = \text{RestrictedExhaustiveSearch}(m, n, S, U, i + 1, D)$ 
   $D2 = \text{RestrictedExhaustiveSearch}(m, n, S, U, i + 1, D \cup \{i + 1\})$ 
  if  $\text{SumIndex}(D1, U) \geq \text{SumIndex}(D2, U)$  then
    return  $D1$ 
  else
    return  $D2$ 
  end if
end if

```

To find a general solution to our problem we simply use the previous method with 0 for i and an empty set as D .

Algorithm 3 `ExhaustiveSearch(m, n, S, U)`

return `RestrictedExhaustiveSearch($m, n, S, U, 0, \emptyset$)`

2.3 Complexity

We will show that the complexity of the method `ExhaustiveSearch` is $O(m2^m)$.

First remark that the auxiliary function `SumIndex` performs $|D| + 2$ computations. Then we prove that the method `RestrictedExhaustiveSearch` executes at most $(m + 5)(2^{m-i} + 2^{m-i+1} - 2)$ instructions if D has size at most i . We reason by induction on $m - i$.

Initialisation. If $m = i$ then we compute `SumIndex(D, S)` in $O(|D|)$, with by assumption $|D| \leq i = m$, and we perform three more operations (two ifs and one return) so the complexity is bounded by $m + 5$.

Preservation. If $m > i$ then we call `RestrictedExhaustiveSearch` twice, with the parameters m , n , S , U , $i + 1$ and either D or $D \cup \{i + 1\}$. By assumption we have $|D| \leq i$ so we have both $|D| \leq i + 1$ and $|D \cup \{i + 1\}| \leq i + 1$ and we can apply the induction hypothesis to both calls, which terminates in at most $(m + 5)(2^{m-i-1} + 2^{m-i} - 2)$ instructions each for a total of $(m + 5)(2^{m-i} + 2^{m-i+1} - 4)$. Moreover the sets $D1$ and $D2$ have at most size m so each calls to `SumIndex` terminates in at most $m + 2$ operations, and we perform three more operations (two ifs and one return, again) for a total of $2m + 7$ additional operations. To conclude we have $(m + 5)(2^{m-i} + 2^{m-i+1} - 4) + 2m + 7 \leq (m + 5)(2^{m-i} + 2^{m-i+1} - 2)$, which is the expected result.

Finally the method ExhaustiveSearch calls RestrictedExhaustiveSearch with $i = 0$ and D empty (hence of size at most 0), so its complexity is $(m + 5)(2^m + 2^{m+1} - 2)$, or $O(m2^m)$.

2.4 Branch and Bound

The basic structure of our branch-and-bound algorithm is the same as the exhaustive search. The difference is that we keep track of the size we use while building the set D , and we make sure never to consider a set whose size exceeds the limit n . The method RestrictedBranchAndBound finds the best solution under the constraint that we already decided whether to pick the databases with index small than i (and this choice is described by D), and n is *the remaining size after picking the elements in D* , which is always kept positive.

Algorithm 4 RestrictedBranchAndBound(m, n, S, U, i, D)

Require: m, n integers, S, U families of integers of size m , $i \leq m$ and D set of integers smaller than i .

Ensure: Returns a set D' of integers between 1 and m such that all $j \leq i$ is in D' if and only if it is in D , $\sum_{j \in D', j > i} S_j \leq n$ and $\sum_{j \in D'} U_j$ is maximal.

if $i = m$ **then**

return D

else

if $S_{i+1} > n$ **then**

return RestrictedBranchAndBound($m, n, S, U, i + 1, D$)

else

$D1 =$ RestrictedBranchAndBound($m, n, S, U, i + 1, D$)

$D2 =$ RestrictedBranchAndBound($m, n - S_{i+1}, S, U, i + 1, D \cup \{i + 1\}$)

if SumIndex($D1, U$) \geq SumIndex($D2, U$) **then**

return $D1$

else

return $D2$

end if

end if

end if

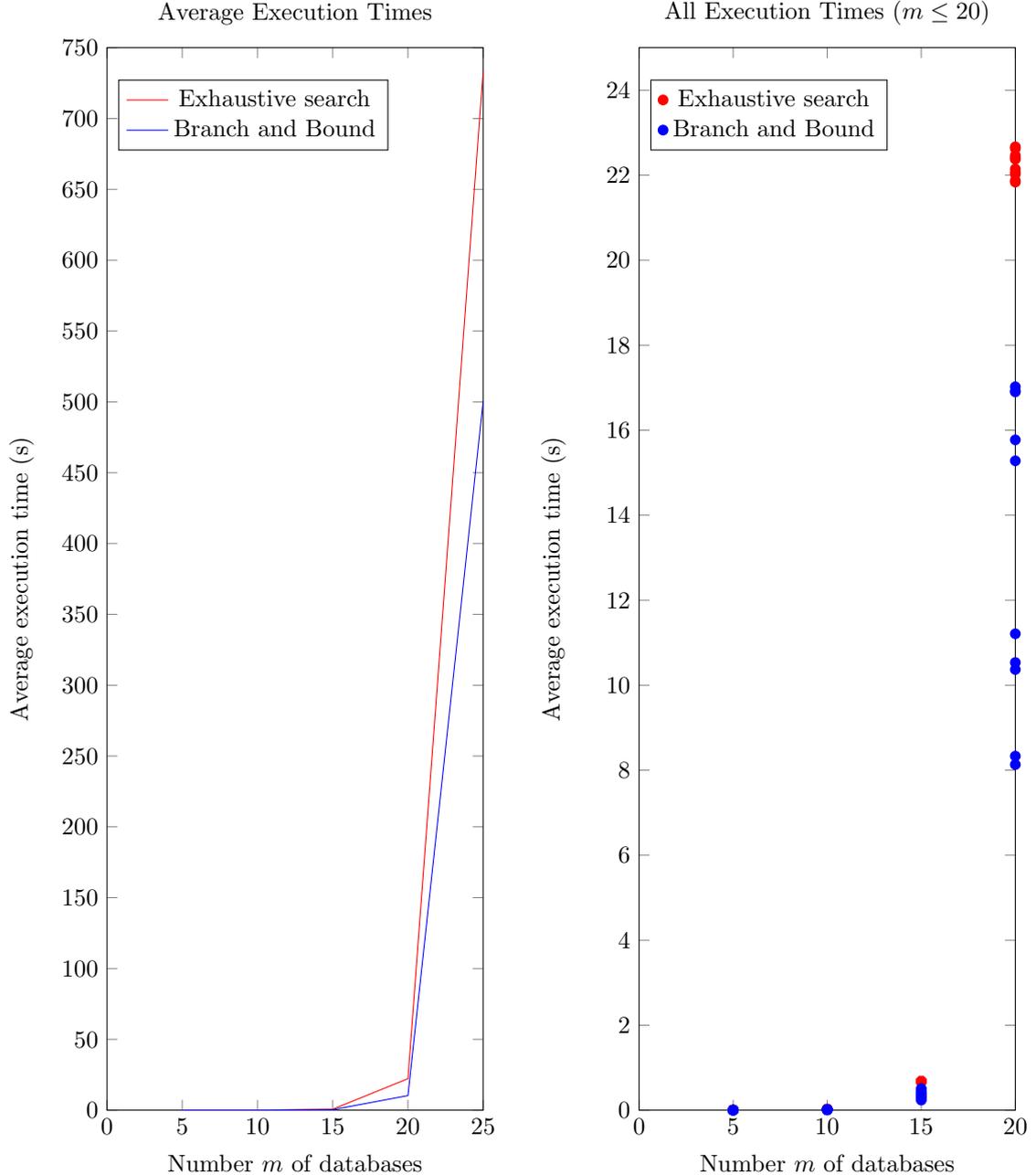
Again the method BranchAndBound just calls the previous method with no constraint.

Algorithm 5 BranchAndBound(m, n, S, U)

return RestrictedBranchAndBound($m, n, S, U, 0, \emptyset$)

2.5 Testing

We present below the experimental execution times of our algorithms when m takes the values 5, 10, 15, 20 and 25, n is $3m$ and S and U are both families of random elements between 1 and 10. The first graph shows the evolution of the average times of 10 executions of each algorithm over the same 10 choices of random families S and U (except for $m = 25$ where each algorithm has been tested only once). The second graph shows all the experimental results.



We can observe that although the branch-and-bound algorithm is faster, it does not appear immensely more efficient than the exhaustive search : it still seems to have an exponential complexity. We can also observe that the execution time of the exhaustive search algorithm does not depend much on the families S and U , whereas the execution time of the branch-and-bound algorithm varies a lot when we run it on different families.