An Exact Algorithm of Polynomial Complexity for the Subset Sum Problem

Andrea Bianchini, Electronics Engineer, andrea@es-andreabianchini.it

4 Marzo 2024

Abstract

The Subset Sum Problem is a well-known NP-complete problem. This paper presents a novel, exact algorithm for the Subset Sum Problem with demonstrably polynomial time complexity. However, the algorithm operates under specific constraints, such as relatively restricted input size compared with (i.e.) web sizes and it’s limited to non negative integer weights. This work contributes by offering an efficient solution for the Subset Sum Problem within these defined boundaries.

1 The Subset Sum Problem

The subset sum problem, a well-known problem in optimization and operations research methods, is defined as follows:

Let c be a positive integer which we will call the capacity of the container.
Let W be a set of n positive integers with a value less than c.

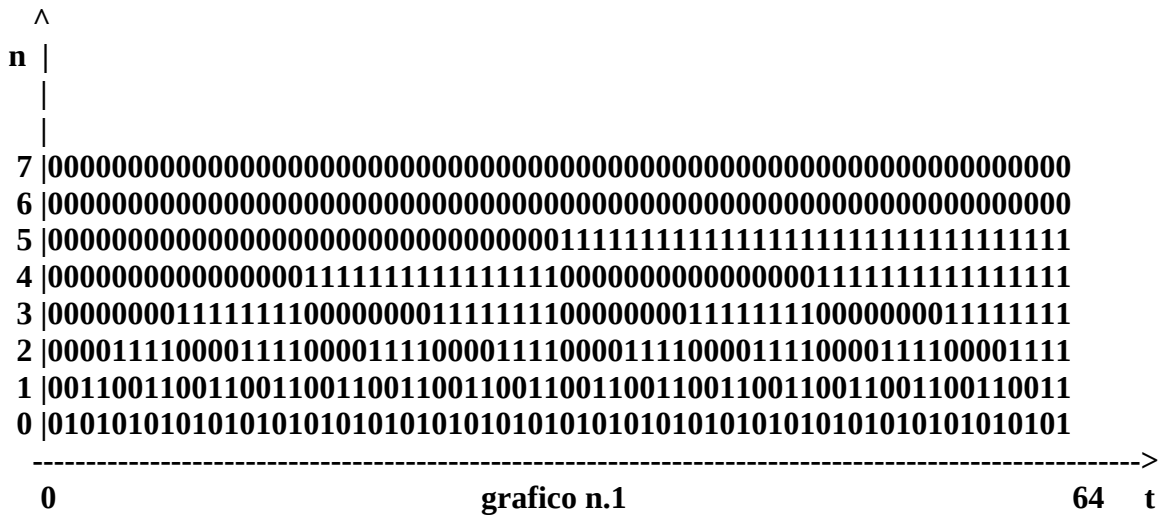
The problem consists in finding, if it exists, a subset of W which we will call Ws such that the sum of the elements of Ws is equal to c.
Mathematically:

find x_i such that $\Sigma x_i * w_i = c$; $w_i < c$; $x_i = 0$ or 1; i = 0,…,n-1; c integer >0; n integer >0;

It has been demonstrated that the subset sum problem is a problem belonging to the class of NP-hard problems, (see “COMPUTERS AND INTRACTABILITY – A Guide to the Theory of NP-Completeness”, Michael R. Garey/David S.Johnson; “ KNAPSACK PROBLEMS” – Silvano Martello/Paolo Toth).

In this article we will illustrate an exact algorithm of polynomial complexity and its proof.

2 Algorithm theory



Graph no. 1 shows on the abscissa the decimal value of the binary number represented by the n bits on the ordinate numbered from the least significant "0" to the most significant "n". In this case n=8 and the bits will be numbered from 0 to 7. As we can clearly observe, the function f(k) (1.0) has a periodic trend with period $T=2^{(k+1)}$. The function g(k) (2.0) where instead of 2^i the value w_i is introduced for i=0, …,k, is also periodic with period $T=2^{(k+1)}$ and the subset sum problem can be solved by the equation g(x)=c. The algorithm presented in this article takes into account the periodicity of g(x) and visits the same bit configurations only once per overall execution, significantly reducing the complexity of the resolution algorithm, obtaining a computational complexity of $O(n^2)$ in the worst case and an average complexity of O(n), as can be easily assessed by observing the algorithm reported in the next paragraph.

$f(k) = \Sigma\, x_i * 2^{i;}$ $x_i=0$ o 1; $i=0,...,k;$ $(T=2^{(k+1)})$ (1.0)

$g(k) = \Sigma\, x_i * w_i;$ $x_i=0$ o 1; $w_i<=w_{i+1};$ $i=0,...,k;$ $(T=2^{(k+1)})$ (2.0)

$g(t)=g(t+T);$ $T=2^{(k+1)};$ $k=0,...,n-1;$ $t=0,...,T;$ (3.0)

find t such that g(t)=C (4.0)

3 The Algorithm

```
def solve():
   global n          # number of items.
   global c          # knapsack capacity
   global w          # item's weights list in ascending order.
   global bestsolu   # list of the best current solution x (binary).
   global sol        # work list of the considered solution x (binary).
   global bestsol    # decimal value of the best currently solution found.


   bestsol=0                                          # variables initialization.
   solvalue=0
   divi=1
   T=int(pow(2,n-1))
   while(T>1 and bestsol<c):                          # first cicle: until T period
      t=0                                             # is greater than  1 o solution
      divi=1                                          # was found. (t=t+int(T/divi)...)

      while(divi<int(pow(2,n)) and bestsol<c):        # second cicle: until the variable
                                                      # divi has scanned all available
         #print(bin(T),bin(t),bestsol)                # subspace or solution was found.


         sol=list(map(int,bin(t+int(T/divi))[2:].zfill(n)[::-1]))   # transform decimal value
         gt=sum([sol[i]*w[i] for i in range(n)])                    # in a zero/one list (binary).
         if (gt<=c):                                                # gt=decimal current solution value.
            t=t+int(T/divi)                                         # if solution is admissible
                                                                    # set bit at 1, (t=t+int(T/divi)).

         divi*=2                                                    # go to the bit
                                                                    # immediately inferior.

         solvalue=gt
         if (bestsol<solvalue and solvalue<=c):                     # if current solution is better
            bestsolu=list(map(int,bin(t)[2:].zfill(n)[::-1]))       # of that found since now,
            bestsol=solvalue                                        # memorize it on list bestsolu.

      T=int(T/2)

   return
```

3.1 Description of the algorithm

The proposed algorithm is written in Python and the main function solve() is provided which at the exit of its execution makes the vector of the optimal solution x available on the binary list bestsolu and its decimal value on bestsol. We can clearly observe the presence of two nested main cycles, the first flows the period T from 2^n to 2 (O(n)), the second searches for a solution for that specific T. In total the solve() function has a complexity of O (n^2) in the worst case, (i.e. there is no optimal solution), and an expected complexity of O(n). For details, please refer to the algorithm and its comments (Paragraph 3).

Possible example of use of the algorithm:

```
inp = open("input.txt","rt")
n=int(inp.readline())
c=int(inp.readline())
wmin=int(inp.readline())
wmax=int(inp.readline())
inp.close()
w=sorted([int(wmin+randrange(wmax-wmin)) for i in range(n)])
sol=[0 for i in range(n)]
bestsolu=[0 for i in range(n)]

print("n="+str(n))
print("c="+str(c))
print("wmin="+str(wmin))
print("wmax="+str(wmax))
print("w = ",w)
print()

bestsol=0

solve()

solvalue=sum([bestsolu[i]*w[i] for i in range(n)])

print()
print("Solution="+str(solvalue))
print("Sol. Vector = ",bestsolu)
print("Sol. Items = ",[w[i] for i in range(n) if bestsolu[i]==1])
```

Special thanks to professors Silvano Martello and Paolo Toth.
Thanks also to Gemini by Google AI Laboratories for helping me write the Abstract and for the psychological support provided to me.