A polynomial-time exact algorithm for the Subset Sum problem

Andrea Bianchini, Electronic/Informatic Engineer, https://www.es-andreabianchini.it

1.0 Definition of the problem.

Subset sum problem (SSP) can be defined as follow :
given a set W of n positive integers and a integer c, (capacity of the knapsack),

find

$$\max z=\sum x(i)w(i) \qquad 1.0$$

s.t.

$$\sum x(i)w(i)\leq c \qquad 1.1$$

$$x(i)=0 \text{ or } 1;\ i=1,\dots,n \qquad 1.2$$

$$0<w(i)\leq c;\quad i=1,\dots,n \qquad 1.3$$

In the present paper it will be always assumed that W is sorted in ascending order, i.e., $w(i+1) \geq w(i)$, $i=0,\dots,n-2$.

Subset sum problem is a well known problem in operations research and it can be proved that it belongs to complexity class NP-Hard, therefore finding an algorithm that solves SSP in polynomial-time prove that P=NP.

1.1 Exploring solutions.

A trivial way to solve SSP is to enumerate all possible binary combination for x and chose the optimal one, requiring in the worst case 2^n iterations.

The basic idea of the presented algorithm derive from the following question :
“does exist a way to explore all binary combination of x in a more efficient way ?”
the answer is : yes it do, and the complexity of this way is polynomial.
Let’s consider the following table that enumerates all binary combination of x for n=5 :

x	base	x	base
00000	5	10000	5
00001	5	10001	5
00010	5	10010	5
00011	2	10011	2
00100	5	10100	5
00101	3	10101	3
00110	3	10110	3
00111	3	10111	3
01000	5	11000	5
01001	4	11001	5
01010	4	11010	5
01011	2	11011	2
01100	4	11100	5
01101	4	11101	5
01110	4	11110	5
01111	4	11111	5

Table 1.0

Definition 1.1.0 : *base* of a binary number

The base of a binary number x is defined by following code :

```
int base(int x[], int n)
{
	int i;

	i=0;
	while(x[i]==0 && i<n)
		i++;
	// i is the position of first "1" bit

	i++;
	// "1" skipped

	while(x[i]==0 && i<n)
		i++;
	// all "0" skipped

	// i is the position of the second "1" bit
	while(x[i]==1 && i<n)
		i++;

	return i;
}
```

As you can see from table 1.0 and from code definition the base of a binary number x is the position of the at least second “1” bit whit successor “0” starting from less significant bit (rightmost bit).

We can obtain all binary numbers of base k starting from 0 adding “1” and shifting one by one until k then again adding “1” and shifting this last until k-1 and so on until all bit from 1 to k are “1”.

Definition 1.1.1 The base k of a binary number x is *pure* if x(i)=0 for all i>k.

Examples for n=5 :

x = 00011	base=2	pure.
x = 10011	base=2	not pure.
x = 00101	base=3	pure.
x = 10101	base=3	not pure.

Cardinality of set of all numbers in a given pure base pb can be easily computed as to be O(∑(pb-i),i=1,…,pb).

Definition 1.1.2 We denote with "x inc k" the increment of x by k positions in the same base of x, and similarly we denote with "x dec k" the decrement of x by k positions in the same base of x.

Examples for n=5 :

x	x inc 1	x dec 1
00001	00010	00000
00100	01000	00010
01100	01101	01010
01101	01110	01100

Proposition 1.1.0 Solutions z=x*w whit all x of the same base are monotone if W is monotone.

Proof. At each increment of x in the given base we add an item w[h] and eventually subtract an item w[k]≤w[h].

Proposition 1.1.1 Searching the maximum of z=x*w not exceeding c in all possible x of the same base can be performed in O(log(n)) time.

Proof. Binary search of a value in a sorted array of values.

1.2 Improving ideas.

Let be “xa” a general feasible solution vector of pure base “ba” and let be “a” the corresponding sum, i.e., a=xa*w.

Proposition 1.2.0 if a≥a’ for all possible a’ with a and a’ of any pure base b=2,…,n, let be ba the base of a, then there exist at least an optimal solution of value osa such that xosa≤xa and xosa>(2^ba)-1, i.e., there exists an optimal solution vector xosa less than or equal to solution vector xa and greater than (2^ba)-1. (base of (2^ba)-1 is ba’=ba-1, therefore grater feasible solution of pure base ba’ is, for definition, less than or equal to a)

Proof. If a=c proof is obvious. Let’s consider a capacity c=a+k, k>0. Let be xosa the optimal solution vector obtainable under condition xosa≤xa and xosa>(2^ba)-1, let be osa it’s solution value, i.e., osa=xosa*w, we can write osa=a+α, α≥0.
We can say that k≥α because osa=a+α≤c, but a=c-k therefore c-k+α≤c therefore k≥α.
Suppose that a solution vector xos>xa or xos≤(2^ba)-1 exists such that os>osa, then we can write os=a’+α’>osa=a+α, but a=c-k therefore a’+α’>c-k+α≥c-k+k, therefore, a’+α’>c that, for definition, is impossible.
It’s important to note that it’s not excluded the presence of an optimal solution xos>xa or xos≤(2^ba)-1, but simply if such solution exists then the same solution value do exists for x≤xa and x>(2^ba)-1.

Proposition 1.2.1 Finding a≥a’ for all possible a’ with a and a’ of any pure base b=2,…,n, can be performed in O(n*log(n)) time.

Proof. It will be shown the O(n*log(n)) algorithm maxABase.

```
int maxABase(int[] w, int n, int c)
{
	int k,k1,lsb1,lsb2,lsbmax,lsbmin,i,amax,basemax,base;

	i=n-1;
	k=0;
	while(k<c && i>=0)
	{
		if (k+w[i]<=c)
		{
			k+=w[i];
			lsb1=i;
		}
		else
			break;
		i--;
	}
	lsbmin=0;
	lsbmax=lsb1-1;
	lsb2=binarySearch(w,c-k,lsbmin,lsbmax);
	if (lsb2>-1)
		k+=w[lsb2];

	amax=0;
	basemax=n;
	base=n;

	while(base>1)
	{
		if (lsb2>-1)
			k-=w[lsb2];
		i=base-1;
		k-=w[i];
		base--;
		i=lsb1-1;
		while(k<c && i>=0)
		{
			if (k+w[i]<=c)
			{
				k+=w[i];
				lsb1=i;
			}
			else
				break;
			i--;
		}
```

```
                lsbmin=0;
                lsbmax=lsb1-1;
                lsb2=binarySearch(w,c-k,lsbmin,lsbmax);
                if (lsb2>-1)
                        k+=w[lsb2];

                if (k>amax)
                {
                        amax=k;
                        basemax=base;
                }
                base--;
        }

        return basemax;
}
```

binarySearch is a function that searches for an item w(lsb2)=max w(i) ≤c-k, i=lsbmin,…,lsbmax, which can be performed in O(log(n)) time.

Proposition 1.2.2 given a≥a' for all possible a' with a and a' of any pure base pb=2,…,n, then finding optimal solution xosa≤xa and xosa>(2^ba)-1, can be performed in O(n^2*log(n)) time.

Proof. We consider SSP', i.e., finding ∑x(i)w(i)≤c-a with items w(i),i<lsb (less significant bit), then SSP'',i.e., finding ∑x(i)w(i)≤c-(a dec 1) with items w(i),i<lsb, until finding ∑x(i)w(i)≤c-(a dec z) with items w(i),i<lsb, where xa dec z is the first available binary number of base a.
Cardinality of set of all numbers in a given pure base pb can be easily computed as to be O(∑(pb-i),i=1,…,pb).

Worst-case time complexity of algorithm.

We can now summarize steps of algorithm for a global worst-case time complexity evaluation :

STEP	COST
1 – Sort of weights array w in ascending order.	O(n*log(n))
2 – Search of max pure base a.	O(n*log(n))
3 – Search of optimal solution vector x≤xa, x>(2^ba)-1.	O(n^2*log(n))

worst case time complexity of algorithm : **O(n^2*log(n))**.
expected time complexity of algorithm : **O(n*log(n))**.

References.

[1] Silvano Martello, PaoloToth, 1990. Knapsack Problems Algorithms And Computer implementations.
[2] Hans Kellerer, Ulrich Pferschy, David Pisinger, 2004. Knapsack Problems.
[3] Michael R. Garey, David S. Johnson WH Freeman, 1979. Computers and Intractability: A guide to the theory of NP-completeness.