First International Conference
Modelling and Development of Intelligent Systems
Sibiu - Romania, 22-25 October, 2009

The Subset Sum Problem

Petko Lalov, Stefan Dimitrov

Abstract

One of the main problems in **Complexity Theory** is the so called **Subset Sum Problem** (**SSP**). Two algorithms for solving this problem are presented in this paper. The first one is polynomial but with approximations. The second one is built on the basis of sequential application of „**Binary Search**", if solution is available, it finds this solution and it has polynomial behaviour.

1 Introduction

One of the major problems in **complexity theory** is the so called **Subset Sum Problem** (**SSP**). Its definition is:
Being given the integer positive number **c** and **n** integer positive numbers $w_1, w_2, \ldots, w_n$, $c \geq w_i$, find:

$$\max \sum_{i=1}^{n} w_i x_i , \tag{1}$$

satisfying the condition

$$\sum_{i=1}^{n} w_i x_i \leq c$$

where $\mathbf{x_i}$ are binary variables.

This problem can also be interpreted in the following equivalent form:

Provided a set of integer positive numbers $W=\{w_1, w_2, \ldots, w_n\}$ is given, does a subset $W' \subseteq W$ exist, such that

$$\sum_{w_i \in A'} w_i = \sum_{w_j \in W \setminus W'} w_j \tag{2}$$

It is evident that the first problem is equivalent to the second if $c = \frac{1}{2} \sum_{i=1}^{n} w_i$.

It is well known that problem (1) can be solved fully exhausting all the possible 2^n variants, represented by n-dimensional binary codes. It is evident that this approach is unacceptable and is applicable if and only if the value of **n** is small enough.
In this paper we suggest several algorithms, of heuristic and precision type, that are extremely effective. The common between them is the considerable reduction of the number of variants considered.

2 Denotations and Definitions

By F_n^2 we denote the set of n-dimensional binary codes. The value of the scalar product **W.x** is denoted by **s(x).** Let **S** is the set of all **s(x)**, where **x** is the binary representation of 1, 2, ...,2^n -1.
In stating the problem we need the function **pos(l,x)** that delivers the corresponding bit (0\1) to position l for code **x**.

Definition 1. If $x_1 \in F_n^2$ *and* $x_2 \in F_n^2$, we say that $x_1 < x_2$ if $x_1^{(10)} < x_2^{(10)}$, where $x^{(10)}$ is the decimal value of x . But from $x_1 < x_2$ it does not always follow that $s(x_1) \le s(x_2)$.

Definition 2. We divide F_n^2 to **n-1** disjunctive sets, called „classes" K_2, K_3, …..,K_n. According [2], $x \in K_l$, l=2, 3, ……,n-1, if the following conditions are satisfied:

- pos(l, x)=1 and pos(l+1, x)=0
- in the positions from 1 to l-1 at least one more bit 1 should exist
- if pos(m, x)=0 for $2 \le m \le l-1$ in positions m-1, m-2, ….1 no more than one bit 1 should be present.

The rest of the codes, that do not satisfy the above conditions, are of the class K_n.

Example: x={1 0 0 1 1 0 1 0} $x \in K_5$
x={0 0 1 0 0 0 1 1} $x \in K_2$
x={1 0 0 0 0 0 0 1} $x \in K_8$

Definition 3. The subset of the class K_l of codes for which $pos(l+2, x)$=pos(l +3,x)=…..=pos(n,x)=0 we call „pure class K_l" and we denote by K_l^p .

From now on we suppose that the set **W** is sorted: $w_n \ge w_{n-1} \ge ..... \ge w_1$.

3 Characteristics of set S

It is not difficult to assume that the solution of (1) is reduced to finding the maximum element of **S** for these **x**, for which $s(x) \le c$. If **S** is sorted, the solution can be found with time complexity of $O(n)$ by the procedure „**binary search**". Unfortunately, in the general case S is not sorted. The single case, for which this is true:

$$w_m \geq w_{m-1} + w_{m-2} + ... + w_1 \quad for\ m = 2,3,....,n$$

Then for each pair $x \in F_n^2$ *and* $y \in F_n^2$ from x<y, it follows that $s(x) \leq s(y)$.

In Figure 1 is drawn the discrete graphics of S at the following data:

W={18, 15, 12, 9, 6, 3, 1}

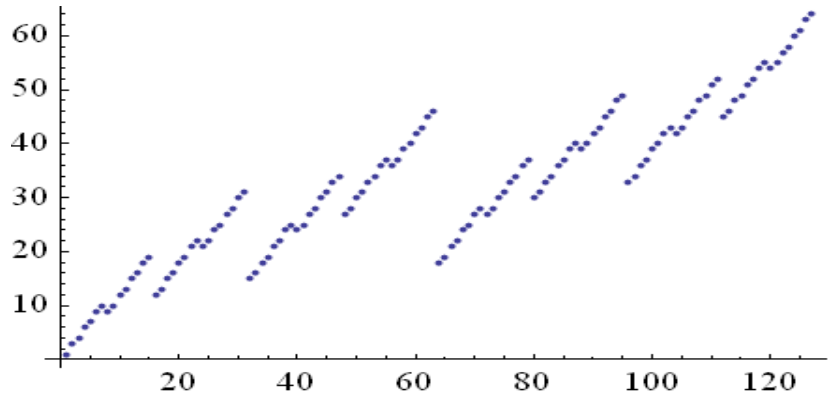


Figure 1

S consists of segments that are sorted. Let define the length and number of these segments. Let **m** is the greatest index for which is true

$$w_m \geq \sum_{i=1}^{m-1} w_i \, , w_{m-1} \geq \sum_{i=1}^{m-2} w_i \, , ..., w_3 \geq \sum_{i=1}^{2} w_i \qquad (3)$$

It is evident that **m** could be at least **2**. By **m** we denote „sorting period".

Theorem 1. The length of each segment of the set **S** at sorting period **m** is $\mathbf{2^m}$.

Proof: The first segment of S includes the values **s(k)**, where **k** are the binary codes of 0, 1, 2,.....,2^m-1, shown in the table that follows:

m	m-1	m-2	……..	3	2	1
0	0	0		0	0	0
0	0	0	…….	0	0	1
0	0	0	…….	0	1	0
0	0	0	…….	0	1	1
0	0	0	…….	1	0	0
……						
1	1	1	……..	1	1	1

It is evident that, considering (3), it is not difficult to assess that if $x^{(10)} < y^{(10)}$ *follows* $s(x) \leq s(y)$.

So **S** can be divided into $\mathbf{2^{n-m}}$ monotonically increasing segments with length 2^m - S_1, S_2,……,$S_{2^{n-m}}$. The limits of these segments are at:

$[s(0), s(2^m - 1)]; [s(2^m), s(2^{m+1} - 1)]; .....[s(2^{n-1}), s(2^n - 1)]$.

The elements of $\mathbf{S_2}$ are obtained from $\mathbf{S_1}$ by adding $\mathbf{w_{m+1}}$ to each element, the elements of $\mathbf{S_3}$ are obtained of $\mathbf{S_2}$ by adding $\mathbf{w_{m+2}}$ to each element, etc.

4 Algorithm A1

1) Q={[0, 2^n-1]}

2) We bisect the interval [0, 2^n-1] and scrutinize the intervals [0, 2^{n-1}-1] and [2^{n-1}, 2^n-1]; determine to which of the intervals [s(0), s(2^{n-1}-1)] and [s(2^{n-1}), s(2^n-1)] belongs **c**. The case **c** to belong to both intervals is not excluded. We form a new set **Q** of these intervals.

3) For each interval of **Q** we apply the bisection procedure and again we form the set **Q**, until the length of intervals of **Q** becomes equal to $\mathbf{2^m}$

4) For each interval, that defines increasing sorted segment, we apply the procedure "binary search" with time complexity $m = Log_2 2^m$ to find maximum element, not surpassing **c** in a sorted array.

4.1 Analysis of A1

The efficiency of this algorithm depends considerably on **m** - „sorting period", and on the value of **c** too. The greater **m** the less is the number of monotonic segments. If we denote with **z** the number of intervals containing **c**, the efficiency of the algorithm is **O(z.m)**.
Surely, the power of **z** can be reached in the worst case $\mathbf{2^{n-m}}$. But in these cases we can restrict ourselves to considering only and only the allowed number of segments. At last, the solution for each segment is allowable and can be considered as a solution, near to the optimal.

5 Algorithm A2

Prior to explaining this algorithm, the classes K_l *and subclass* K_l^p are considered. Let find the powers of these sets. These are the codes of K_l^p:

0….0 1 0 0… 0 0 1
0….0 1 0 0… 0 1 0
……..
0….0 1 1 0… 0 0 0
0….0 1 1 0… 0 1 0
……
0….0 1 1 1… 0 0 0
……
0….0 1 1 1… 1 1 1
l+1 l l-1.....1

It is not difficult to compute that $|K_l^p| = \frac{l(l-1)}{2}, l = 2,..,n-1, |K_n^p| = \frac{n(n+1)}{2}, a\ |K_l| = 2^{n-l-1}.\frac{l(l-1)}{2},\ l = 2,...,n-1, |K_n| = \frac{n(n+1)}{2}$

For the classes considered we will prove the following theorem:

Theorem 2. *If* $x, y \in K_l^p$ *and* $x < y$ *then* $s(x) \le s(y)$

Proof: Let $x < y$. This means that for j<l

$$pos(l,x) = pos(l,y) = 1, pos(l-1,x) = pos(l-1,y) = 1, ..., pos(l-j,x) = pos(l-j,y) = 1,$$
$$and\ \ pos(l-j-1,x) = pos(l-j-1,y) = 0$$

Therefore, if the position of next remaining single **1** for **x** is **p**, and for **y** is **q**, then it is evident that pos[q,y]>pos[p,x] and $w_p \le w_q$, from which we can conclude that

$$s(x) = \sum_{i=l-j}^{l} w_i + w_p,\ s(y) = \sum_{i=l-j}^{l} w_i + w_q \ \text{ or } \ s(x) \le s(y)$$

The theorem just proved, it follows that the set $\{s(x) \mid x \in K_l^p\}$ is sorted for each l=2,3,….,n.

The solution of (1) we search sequentially in „pure classes" $K_2^p, K_3^p, ..., K_n^p$, that are sorted. And that makes the algorithm heuristic, i.e. the solution is not always optimal but is very close to optimal. Computer-based experiments confirmed the latter conclusion. On finding the best „pure" solution, the latter can be improved if necessary. That is the short description of A2:

1) $x_l = \max\{s(x), x \in K_l^p\}, l = 2,3,...,n$

2) $x_l = \max\{x_2, x_3, ..., x_n\};\ s_{\max} = s(x_l)$

3) if $s_{\max} = c \rightarrow Stop$,else 4)

4) for $x < x_l, x \in K_l^p$ we search $\max\{w_{l+2}, w_{l+3}, ..., w_n\}$, so that

$$w_i \le c - s_{\max},\ w_i \in \{w_{l+2}, w_{l+3}, ..., w_n\}$$

if we find solution in the i-bit of $x, i \in \{l+2, l+3, ..., n\}$ we insert 1, hence $x \in K_l \setminus K_l^p$. For the new **x** we apply 4). i.e. we search $\max\{w_{i-1},\ w_{i-2}, ..., w_{l+2}\}$ so that

$$w_j \le c - s_{\max} - w_i,\ w_j \in \{w_{i-1},\ w_{i-2}, ..., w_{l+2}\} \text{ etc.}$$

Step 4) is performed using "binary search", applying an idea from [2].

5.1 Analysis of A2

Considering that $|K_l^p| = \frac{l(l-1)}{2}, l = 2,3,....., n-1, |K_n^p| = \frac{n(n+1)}{2}$ finding out $s_{\max}$ has time complexity $O(\sum_{k=2}^{n-1} \log_2 \frac{k(k-1)}{2})$, which is in the order of magnitude of $\ln(G[n]G[n-1])$, where G is the well known special Euler function. Even if we take for granted that each class K_l^p is with length **n**, processing **n** sorted classes will take time much less than $2n\log_2 n$ or

$$\ln(G[n]G[n-1]) < 2n\log_2 n$$

In Figure 2 are shown graphics of the two functions for n≤5000.

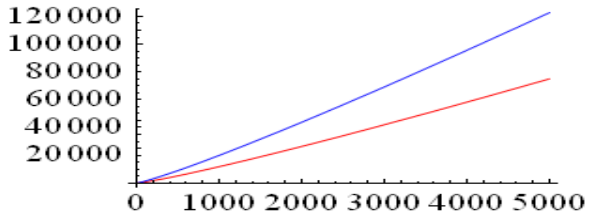


Figure 2

6 Algorithm A3

This algorithm is a result of the application of the so called „greedy algorithm" that is implemented by repeatedly applying the "binary search" and backtracking. Step-by-step description of the algorithm follows:

1) $w_{i_1} = \max\{w_n, w_{n-1}, ..., w_1\} \le c$

$w_{i_2} = \max\{w_{i_1-1}, w_{i_1-2}, ..., w_1\} \le c - w_{i_1}$

……

$w_{i_k} = \max\{w_{i_{k-1}-1}, w_{i_{k-1}-2}, ..., w_1\} \le c - w_{i_1} - w_{i2} - ... - w_{i_{k-1}}$

2) code $\mathbf{p_1}$ is generated with $pos(i_1, p_1) = 1; pos(i_2, p_1) = 1; .....; pos(i_k, p_1) = 1$

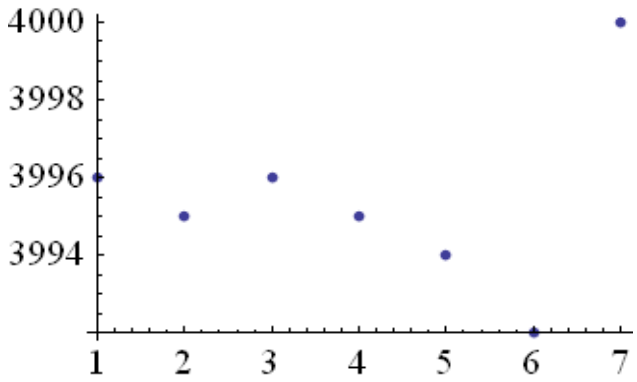
3) if $w_{i_1} + w_{i_2} + ... + w_{i_k} = c \rightarrow Stop$

4) $pos(i_k, p_1) = 0$

5) steps 1), 2), 3) are repeated for the set $\{w_{i_k-1}, w_{i_k-2}....., w_1\}$ and $c = c - w_{i_k}$

At last, a sequence of codes $\mathbf{p_1, p_2, \ldots}$ is obtained that reaches to the solution.

In the graphics below is shown how it can be reached to the solution of (1) passing through 7 iterations for an instance of **n=50** , **c=4000** and $\mathbf{w_i}$ generated in random.



7 Conclusion

Each of the algorithms presented has its advantages and disadvantages.
In algorithm A1 the original idea is the bisection of set S to sorted segments and search for the solution in a part of these segments. This idea is subject to future development.
Algorithm A2 has the most effective time complexity, when searching for a solution in the so called „pure" sorted classes $K_2^p, K_3^p, ..., K_n^p$, but it not always yields optimal result, no matter how close to it. This conclusion is confirmed by the numerical experiments made.
Algorithm A3 is a combination of „greedy algorithm", "binary search" and „backtracking". It always yields the optimal solution, but it is very difficult to evaluate its efficiency. The numerical experiments demonstrated polynomial complexity when $\max \sum_{i=1}^{n} w_i x_i = c$.

Reference:

[1] M. R. Garey, D. S. Johnson, *Computers and intractability. A Guide to the Theory of NP- Completeness*, San Francisco 1979, Moscow 1982

[2] Andrea Bianchini, *A Polynomial-time Exact Algorithm for the Subset Sum Problem*, Engineer, www.bianchiniandrea.it

Petko Lalov
University of Mining and Geology "St. Ivan Rilski"
Head of chair of Informatics
Students' Town, Sofia
BULGARIA
E-mail: petko@mgu.bg

Stefan Dimitrov
University of Sofia "St. Kliment Ohridski"
University Computer Centre
5, J. Bourchier, Str., Sofiq
BULGARIA
E-mail: stefan@ucc.uni-sofia.bg