# Polynomial-time Matrix-based Method of Determining Subset Sum Solutions

Aubrey Alston

**Abstract**

Reducing the conditions under which a given set satisfies the stipulations of the subset sum proposition to a set of linear relationships, the question of whether a set satisfies subset sum may be answered in a polynomial number of steps by forming, solving, and constraining a series of linear systems whose dimensions and number are both polynomial with respect to the length of the set. Given its demonstrated 100% accuracy rate and its demonstrable justification, this algorithm may provide basis to reconsider the validity of SSP-based and NP-complete-reliant cryptosystems.

## 1 INTRODUCTION

The subset sum problem is a well-known member of the NP-complete complexity class: given a set of integers $A$ and some constant $c$, is there some subset of $A$ which sums to $c$?

Previously, there have been no known algorithms or methods to solve the value-unbounded, general-case subset sum problem in polynomial time. The naive algorithm performs in exponential time by cycling through the possible subsets of $A$ until it has either seen all subsets or found one which sums to $c$. A common pseudo-polynomial method employs a dynamic programming algorithm whose complexity is polynomial with respect to the length of the set and the range of the inputs, $O(n(M\text{-}N))$, where n is the length of the set and $B\text{-}A$ is the range of inputs; however, this solution is not truly polynomial, as it is polynomial with respect to $M\text{-}N$, which is exponential in its number of bits. Approximate algorithms exist which can be modified to find exact solutions; however, they too degrade to being exponential in the number of bits required to represent elements in the set.

In contrast with pre-existing algorithms, the method described here does not concern itself with the various subsets that exist within the input set, but rather searches the solution space of a set of linear constraints when applied to an input set to deduce if a solution can exist; the method to be discussed is a strategy which may be employed to find solutions satisfying the constraints of the subset sum problem in time polynomial with respect only to the length of the input, having general-case applicability on the basis of universally occuring properties in sets satisfying the problem.

1

# 2 Method

## 2.1 Preliminary Conventions, Definitions, and Properties

Given a set $A$ of $n$ greater than four elements and an instance of subset sum for a constant $c$, satisfied by a subset $S$ of length greater than 2 (the algorithm first catches trivial cases for S of length 1 or 2), index A as follows:

$$A = \{a_1, a_2, ..., a_n\} \tag{1}$$

The strategy outlined will conform $A$ to a set of linear constraints which will reveal a subset sum-satisfying subset if one exists. To this end, define a subset membership vector $m$ specific to $A$ such that the $i$th value in $m$ is 1 if the $i$th element of $A$ is an element of a given subset $S$ of $A$ summing to $c$, 0 otherwise. If such a subset $S$ exists, $m$ exists and encodes $S$ within $A$.

$$m = \begin{pmatrix} m_1 \\ \vdots \\ \vdots \\ m_n \end{pmatrix} \tag{2}$$

If $S$ exists, the four following linear constraints surrounding $A$, $S$, and $m$ will be satisfied:

1. $S$ sums to $c$.
$$a_1 m_1 + a_2 m_2 + ... + a_n m_n = c \tag{3}$$

2. $S$ has finite length $t$.
$$m_1 + m_2 + ... + m_n = |S| = t \tag{4}$$

3. There exists an index $r$ for which the $r$th element of $A$ is or is not in $S$.
$$m_r = v \in \{0,1\} \tag{5}$$

4. There exists an index $s$ for which the $s$th element of $A$ is or is not in $S$.
$$m_s = v \in \{0,1\} \tag{6}$$

Using the above four constraints, an underdetermined system $Z$ can be constructed in the parameters of the constraints listed:

$$\begin{aligned} Z(A, t, r, v_r, s, v_s) : \\ a_1 m_1 + a_2 m_2 + ... + a_n m_n = c \\ m_1 + m_2 + ... + m_n = t \\ m_r = v_r \\ m_s = v_s \end{aligned} \tag{7}$$

The algorithm given below explores the solution spaces of a polynomial number of forms of Z to construct the characteristic membership vector $m$ for some subset $S$ of $A$ summing to $c$ if one exists. The convention for the determining solution space of $Z(A, t, r, v_r, s, v_s)$ is to first form an equivalent set representation $A'$ by interchanging index 3 of a with index r, interchanging index 4 with index s in $A$, and solving $Z(A', t, 3, v_r, 4, v_s)$ for $m'$ equal to $m$ with likewise index permutations using matrices.

$$
\begin{pmatrix}
a_1 & a_2 & a_r & a_s & \cdots & a_n \\
1 & 1 & 1 & 1 & \cdots & 1 \\
0 & 0 & 1 & 0 & \cdots & 0 \\
0 & 0 & 0 & 1 & \cdots & 0
\end{pmatrix}
\begin{pmatrix}
m_1 \\ m_2 \\ m_r \\ m_s \\ \vdots \\ m_n
\end{pmatrix}
=
\begin{pmatrix}
c \\ t \\ v_r \\ v_s
\end{pmatrix}
\tag{8}
$$

To represent the solution space of $Z(A', t, 3, v_r, 4, v_s)$, the outlined algorithm follows the convention of expressing solution space with respect to a particular solution of the system and any linear combination of the null space of the multiplier of $Z(A', t, 3, v_r, 4, v_s)$.

$$
m' = m'_p + dN(
\begin{pmatrix}
a_1 & a_2 & a_r & a_s & \cdots & a_n \\
1 & 1 & 1 & 1 & \cdots & 1 \\
0 & 0 & 1 & 0 & \cdots & 0 \\
0 & 0 & 0 & 1 & \cdots & 0
\end{pmatrix}
)
\tag{9}
$$

The followed solution convention is simple forward elimination, followed by back-substitution. The particular solution is chosen such that all free variables (the rank of the multiplier of the system is 4; given its representation, the free variables are $a'_5, \ldots a'_n$) are assumed to be zero. The null space is then composed of $n$-4 special solutions each respectively assuming one unique $m_i$, $i = 5, \ldots, n$ to be 1, all other $m_j$ to be 0. This convention then allows $m'$ for any $A'$ to be expressed as follows:

$$
m' =
\begin{pmatrix}
b_1 \\ b_2 \\ v_r \\ v_s \\ 0 \\ \vdots \\ 0
\end{pmatrix}
+ d_1
\begin{pmatrix}
k_{1,1} \\ k_{2,1} \\ 0 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0
\end{pmatrix}
+ d_2
\begin{pmatrix}
k_{1,2} \\ k_{2,2} \\ 0 \\ 0 \\ 0 \\ 1 \\ \vdots \\ 0
\end{pmatrix}
+ \ldots + d_{n-4}
\begin{pmatrix}
k_{1,n-4} \\ k_{2,n-4} \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1
\end{pmatrix}
\tag{10}
$$

The prescribed convention thus provides a representation of the solution space that may be explored to determine if a valid $m'$ satisfying subset sum exists for a given $Z(A', t, 3, v_r, 4, v_s)$: each vector of the null space can be contributed to $m'$ one or zero times, and, together, the sum of the particular

solutions and applicable vectors of the null space will take $b_1$ and $b_2$ (the non-zero and non-one values in the particular solution unique to the values of the first two elements and the chosen $v_r$ and $v_s$) to 0 or 1. For any solution space for which this applies, the resulting $m'$ will be a vector of 0s and 1s encoding the membership of $S$. Due to the form found as a result of convention of this method, the first four elements of $A'$ are referred to as the *current window*, the first two elements are the *balance elements*, and the second two elements are the *pivot elements*.

The algorithm below attempts to reveal and exploit solution space properties which may exist universally among all sets for **some** window configuration if the set satisfies subset sum to form $m'$. If the found properties are, in fact, universal, the algorithm outlined is an exact, general-case algorithm for the subset sum problem.

In order to determine, view, and exploit these properties, the algorithm utilizes a construct which will be called a *directional contribution table*. A directional contribution table is a tabulation of the contributions of the elements of the null space towards bringing the balance values of the particular solution towards 0 or 1. A directional contribution table $D$ tabulates the contribution of a given vector in the null space of $Z$ towards paired balance targets and is defined with respect to the solution space (as expressed in the previously given convention) of a system $S(Z)$ and given target values of the balance points within the particular solution.

$$D(S(Z), t_1, t_2):$$

$$\begin{bmatrix} \frac{k_{1,1}}{t_1-b_1} & \frac{k_{1,2}}{t_1-b_1} & \cdots & \frac{k_{1,n-4}}{t_1-b_1} \\ \frac{k_{2,1}}{t_2-b_2} & \frac{k_{2,2}}{t_2-b_2} & \cdots & \frac{k_{2,n-4}}{t_2-b_2} \\ \frac{k_{1,1}}{t_1-b_1}-\frac{k_{2,1}}{t_2-b_2} & \frac{k_{1,2}}{t_1-b_1}-\frac{k_{2,2}}{t_2-b_2} & \cdots & \frac{k_{1,n-4}}{t_1-b_1}-\frac{k_{2,n-4}}{t_2-b_2} \end{bmatrix} \tag{11}$$

For a given set $A$ satisfying subset sum, there appears to exit a window configuration for which in $Z(A', t, 3, v_r, 4, v_s)$ and $D(S(Z), t_1, t_2)$, $t$, $v_r$, $v_s$, $t_1$, and $t_2$ apply to an extant $S$, characterized by specific properties within $D$. The following (possibly non-exhaustive) property has been determined and is employed by the algorithm to determine $m'$ encoding $S$ within $A'$:

(A) If the length of $S$ is 4, $m'$ is the exact solution of S(Z) when the elements of the window are the elements of $S$ and membership variables are set appropriately. If the length of $S$ is 5, $m'$ is the exact solution of S(Z) plus the vector representing the column of D for which $D_{1,i}$ and $D_{1,i}$ are both 1 when membership variables are set appropriately. In all other cases for a set $A$ of length greater than four, $m'$ may be formed by taking the vectors represented by each column $i$ for which the absolute value of $D_{3,i}$ is less than one.

## 2.2 Justification of Properties

Within the parameters of the convention listed above, manual algebraic reduction in the general case yields the following closed forms for the variables of

the particular solution of S(Z), the null space of S(Z), and values within the directional contribution table:

$$\beta = a_1 - a_2 \tag{12}$$

$$b_1 = \frac{c - v_s a_s - v_r a_r - a_2(t - v_r - v_s)}{\beta} \tag{13}$$

$$b_2 = \frac{a_1(t - v_r - v_s) + v_s a_s + v_r a_r - c}{\beta} \tag{14}$$

$$k_{1,i} = \frac{a_2 - a_{i+4}}{\beta} \tag{15}$$

$$k_{2,i} = \frac{a_{i+4} - a_1}{\beta} \tag{16}$$

$$\delta_1 = t_1(a_1 - a_2) + v_s(a_s - a_2) + v_r(a_r - a_2) + a_2 t - c \tag{17}$$

$$\delta_1 = a_2(t - t_1 - v_r - v_s) - (c - t_1 a_1 - v_r a_r - v_s a_s) \tag{18}$$

$$\delta_2 = t_2(a_1 - a_2) - v_s(a_s - a_1) - v_r(a_r - a_1) - a_1 t + c \tag{19}$$

$$\delta_2 = (c - t_2 a_2 - v_r a_r - v_s a_s) - a_1(t - t_2 - v_r - v_s) \tag{20}$$

$$D_{1,i} = \frac{a_2 - a_{i+4}}{\delta_1} \tag{21}$$

$$D_{2,i} = \frac{a_{i+4} - a_1}{\delta_2} \tag{22}$$

$$D_{3,i} = \left| \frac{\delta_1 a_1 + \delta_2 a_2 - a_{i+4}(\delta_1 + \delta_2)}{\delta_1 \delta_2} \right| \tag{23}$$

Property (A) is to say that an instance of a subset $S$ of $A$ summing to $c$ exists under the following constraints:

(1) The length of $S$ is four, and when the elements of $S$ are set as the window of $A'$, $m'$ is found encoding $S$ within $A$.

Assume all members of $S$ are the current window of $A'$, and set membership to $t_1 = 1$, $t_2 = 1$, $v_r = 1$, $v_s = 1$.

$$c = a_1 + a_2 + a_r + a_s$$

$$b_1 = \frac{c - a_s - v_r - 2a_2}{a_1 - a_2}$$

$$b_1 = \frac{a_1 + a_2 - 2a_2}{a_1 - a_2}$$

$$b_1 = 1$$

$$b_2 = \frac{2a_1 + v_r + v_s - a_1 - a_2 - v_r - v_s}{a_1 - a_2}$$

$$b_2 = \frac{a_1 - a_2}{a_1 - a_2}$$

$$b_2 = 1$$

Thus, the particular solution of S(Z) for this configuration is

$$\begin{pmatrix} b_1 \\ b_2 \\ m_r \\ m_s \\ \vdots \\ m_n \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ \vdots \\ 0 \end{pmatrix}$$

encoding $S$ and showing (1) to be true. The above also extends to show that (1) is true for any instance of $S$ having length less than four for which all elements of $S$ are within the window of $A'$ and membership is set appropriately.

(2) The length of S is 5, and when all but one element of $S$ is set as the window of $A'$, $m'$ encoding $S$ within $A'$ is found by adding to the particular solution the vector of the null space corresponding to $D_{1,r} = D_{2,r} = 1$.

Assume four members of $S$ are the current window of $A'$, and set membership to $t_1 = 1$, $t_2 = 1$, $v_r = 1$, $v_s = 1$.

$$c = a_1 + a_2 + a_r + a_s + a_k$$

Solving for the values in the directional contribution table:

$$D_{1,k-4} = \frac{a_2 - a_k}{a_1 a_4 + a_3 + 2a_2 - c}$$

$$D_{1,k-4} = \frac{a_2 - a_k}{a_2 - a_k} = 1$$

$$D_{2,k-4} = \frac{a_k - a_1}{c - a_2 - a_4 - a_3 - 2a_1}$$

$$D_{2,k-4} = \frac{a_k - a_1}{a_k - a_1} = 1$$

showing (2) to be true. The above may also be generalized for any S having length less than or equal to five for which all but one element of the subset exists within the window and membership is set appropriately.

(3) In all other cases, $S$ of length $t > 5$ of $A$ exists if and only if there exists a window configuration and membership assignment $t_1$, $t_2$, $v_r$, $v_s$ such that there

exists a set $I$, length $t' = t - t_1 - t_2 - v_r - v_s$ satisfying $D_{3,I_j} < 1$ and $\Sigma D_{1,I_j} = \Sigma D_{2,\,I_j} = 1$ for all $j$ in the range of the length of $I$.

Suppose that such a window exists. It is given that

$$\sum_{j=1}^{t'} D_{1,I_j} = 1$$

and

$$\sum_{j=1}^{t'} D_{2,I_j} = 1$$

Using the closed forms derived earlier in this section:

$$\sum_{j=1}^{t'} D_{1,I_j} = 1 = \sum_{j=1}^{t'} \frac{K_{1,I_j}}{t_1 - b_1}$$

$$t_1 - b_1 = \sum_{j=1}^{t'} K_{1,I_j}$$

$$b_1 + \sum_{j=1}^{t'} K_{1,I_j} = t_1$$

$$\sum_{j=1}^{t'} D_{2,I_j} = 1 = \sum_{j=1}^{t'} \frac{K_{2,I_j}}{t_2 - b_2}$$

$$t_2 - b_2 = \sum_{j=1}^{t'} K_{2,I_j}$$

$$b_2 + \sum_{j=1}^{t'} K_{2,I_j} = t_2$$

Repeating the same process as was used in the justification of (1), $m'$ is derived encoding $S$ within $A'$:

$$m' = x_p + \sum_{j=1}^{t'} N(A)_{I_j} = \begin{pmatrix} t_1 \\ t_2 \\ v_r \\ v_s \\ \vdots \end{pmatrix} \quad m_k, k > 4 = \begin{cases} 1, & \text{if } k - 4 \in I \\ 0, & \text{if } k - 4 \notin I \end{cases}$$

$$S = \left\{ A'_k \mid m'_k = 1 \right.$$

$$\mid S \mid = t_1 + t_2 + v_r + v_s + t' = t$$

7

Thus, if such a window exists, a $S$ of $A'$ of length $t$ exists and is encoded by the derivable $m'$.

Suppose $S$ of length $t$ of $A'$ exists. Let

$$S = \{s_1, s_2, ..., s_t\}$$

$$S' = A\backslash S = \{a_l \mid a_l \notin S\}$$

$$I = \{i - 4 \mid A'_i \in S, i > 4\}$$

$$X = \{i - 4 \mid A'_i \notin S, 4 < i < max(I)\}$$

Call window $W = \{\, A'_1,\ A'_2,\ A'_3,\ A'_4 \,\}$ and define membership of the window as follows:

$$t_1 = \begin{cases} 1, & \text{if } w_1 \in S \\ 0, & \text{if } w_1 \notin S \end{cases} \quad t_2 = \begin{cases} 1, & \text{if } w_2 \in S \\ 0, & \text{if } w_2 \notin S \end{cases}$$

$$v_r = \begin{cases} 1, & \text{if } w_3 \in S \\ 0, & \text{if } w_3 \notin S \end{cases} \quad v_s = \begin{cases} 1, & \text{if } w_4 \in S \\ 0, & \text{if } w_4 \notin S \end{cases}$$

Under this membership assignment,

$$\sum_{j=1}^{t'} D_{1,I_j} = \sum_{j=1}^{t'} \frac{a_2 - a_{I_j+4}}{\delta_1} = \frac{1}{\delta_1}\left(t'a_2 - \sum_{j=1}^{t'} a_{I_j+4}\right)$$

$$= \frac{1}{\delta_1}(a_2 t - a_2 t_2 - a_2 v_r - a2 v_s - (c - a_1 t_1 - a_2 v_2 - a_r v_r - a_s v_s))$$

$$= \frac{1}{\delta_1}(t_1(a_1 - a_2) + v_s(a_s - a_2) + v_r(a_3 - a_2) + a_2 t - c)$$

$$= 1$$

Likewise,

$$\sum_{j=1}^{t'} D_{2,I_j} = \sum_{j=1}^{t'} \frac{a_{I_j+4} - a_1}{\delta_2}$$

$$= \frac{1}{\delta_2}(c - t_1 a_1 - a_2 t_2 - v_r a_r - v_s a_s - a_1 t + a_1 t_1 + a_2 t_2 + a_r v_r + a_s v_s)$$

$$= 1$$

Thus, if $S$ of $A$ exists, $\Sigma D_{2,I_j} = \Sigma D_{1,I_j} = 1$.

By (3), all $D_{3,I_j}$ for the window must satisfy

$$-1 < \frac{\delta_1 a_1 + \delta_2 a_2 - a_{I_j+4}}{\delta_1 \delta_2} < 1$$

Expanding the variables and simplifying, for $\delta_1\delta_2 > 0$,

$$a_{I_j+4}(\delta_1+\delta_2) - \delta_1\delta_2 < (a_2-a_1)(c - t_1 a_1 - t_2 a_2 - v_r a_r - v_s a_s) < a_{I_j+4}(\delta_1+\delta_2) + \delta_1\delta_2$$

8

and for $\delta_1\delta_2 < 0$,

$$a_{I_j+4}(\delta_1+\delta_2)+\delta_1\delta_2 < (a_2-a_1)(c-t_1a_1-t_2a_2-v_ra_r-v_sa_s) < a_{I_j+4}(\delta_1+\delta_2)-\delta_1\delta_2$$

It can be shown that the above is satisfied for some window configuration by showing that a tighter contained constraint is also satisfied for some window:

$$-t' < \sum_{j=1}^{t'} D3, I_j < t',$$

$$D_{1,I_j} > 0 \text{ and } D_{2,I_j} > 0$$

Summing the $D_{3,I_j}$,

$$\sum_{j=1}^{t'} D_{3,I_j} = \frac{1}{\delta_1\delta_2} \sum_{j=1}^{t'} (\delta_1 a_1 + \delta_2 a_2 - a_{I_j+4}(\delta_1 + \delta_2))$$

$$= t'\left(\frac{a_1\delta_1 + a_2\delta_2}{\delta_1\delta_2}\right) - \frac{c - t_1a_1 - t_2a_2 - v_ra_3 - v_sa_4}{\delta_1\delta_2}$$

$$= \frac{t'(a_1 - a_2)(t_1a_1 + t_2a_2 + v_ra_r + v_sa_s - c) + (t_1a_1 + t_2a_2 + v_ra_r + v_sa_s + c)}{\delta_1\delta_2}$$

$$= \frac{(t_1a_1 + t_2a_2 + v_ra_r + v_sa_s - c)(t'(a_1 - a_2) + 1)}{\delta_1\delta_2}$$

$$= \frac{(t_1a_1 + t_2a_2 + v_ra_r + v_sa_s - c)(t'a_1 - t'a_2 + 1)}{(a_2t' + a_1t_1 + a_rv_r + a_sv_s - c)(c - a_1t' - a_2t_2 - a_rv_r - a_sv_s)}$$

$$= \frac{(c - t_1a_1 - t_2a_2 - v_ra_r - v_sa_s)(t'a_1 - t'a_2 + 1)}{(c - a_2t' - a_1t_1 - a_rv_r - a_sv_s)(c - a_1t' - a_2t_2 - a_rv_r - a_sv_s)}$$

The numerator and denominator of this expression are both polynomial with respect to $t'$ (the numerator is of degree 1, and the denominator of degree 2); given that $t' > 1$, all coefficients of $t'$ in the numerator appear as a product in the denominator, and all non-$t'$ constants in the numerator appear in the denominator, the result is bounded by (-t',t').

Given that the above is bounded, vewing the second constraint, assert that

$$D_{1,I_j} > 0 \text{ and } D_{2,I_j} > 0$$

Thus

$$\frac{a_2 - a_{I_j+4}}{\delta_1} > 0 \text{ and } \frac{a_{I_j+4} - a_1}{\delta_2} > 0$$

meaning

$$\begin{cases} a_2 > a_{I_j+4}, & \text{if } \delta_1 > 0 \\ a_2 < a_{I_j+4}, & \text{if } \delta_1 < 0 \end{cases} and \begin{cases} a_1 < a_{I_j+4}, & \text{if } \delta_2 > 0 \\ a_1 > a_{I_j+4}, & \text{if } \delta_2 < 0 \end{cases}$$

9

Noting that

$$\begin{cases} \delta_1 > 0, & \text{if } a_2 > \frac{c - t_1 a_1 - v_r a_r - v_s a_s}{t - t_1 - v_r - v_s} \\ \delta_1 < 0, & \text{if } a_2 < \frac{c - t_1 a_1 - v_r a_r - v_s a_s}{t - t_1 - v_r - v_s} \\ \delta_2 > 0, & \text{if } a_1 < \frac{c - t_2 a_2 - v_r a_r - v_s a_s}{t - t_2 - v_r - v_s} \\ \delta_2 < 0, & \text{if } a_1 > \frac{c - t_2 a_2 - v_r a_r - v_s a_s}{t - t_2 - v_r - v_s} \end{cases},$$

it can be seen that a window configuration composed of the minima and/or maxima (and/or points whose values are set according to these) with membership appropriately set of $s_i$ can be made among the set *(or possibly created)* to satisfy both these (and thus the primary) constraints.

Additionally, note that if an initial ordering of the set is enforced in which the set is ordered by the absolute value of its elements (increasing or decreasing), enumerating and swapping all possible windows of $A$ in a pair-wise order (1 with 2, 1 with 3, . . ., 2 with 3; followed by pair 1 with pair 2 .. . . pair 2 with pair 3) creates such a set ordering for which a window may satisfy

$$D_{3,X_k} > 1 \text{ or } D_{3,X_k} < -1$$

in the same window assignment as the previous constraint.

This shows, then, that if $S$ of length $t$ of $A$ exists, there exists a window configuration and membership assignment $t_1$, $t_2$, $v_r$, $v_s$ such that there exists a set $I$, length $t' = t - t_1 - t_2 - v_r - v_s$ satisfying $D_{3,I_j} < 1$ and $\Sigma D_{1,I_j} = \Sigma D_{2, I_j} = 1$ for all $j$ in the range of the length of $I$.

## 2.3   Algorithm

Subset Sum Algorithm. Input: set of constants A and constant c. Output: non-empty subset S of A summing to c if such a subset exists.

```
 1: procedure SUBSETSUM(A,c)
 2:     add to A a value fabricated according to constraint properties
     ▷ TEST and PARTIALSUBSETSUM procedures should not return any
     subset containing this value.
 3:     sort A by descending absolute value
 4:     for length(A) iterations do
 5:         S ← PARTIALSUBSETSUM(A, c)
 6:         if length(S) >0 then
 7:             return S
 8:         end if
 9:     end for
10:     sort A by ascending absolute value
11:     for length(A) iterations do
12:         S ← PARTIALSUBSETSUM(A, c)
13:         if length(S) >0 then
14:             return S
15:         end if
16:     end for
```

17:      return empty list
18: **end procedure**


19: **procedure** PARTIALSUBSETSUM($A, c$)
20:      Initialize empty list $S$.
21:      Initialize empty list *pairs*.
22:      Initialize empty list *windows*.
23:      $runningSum \leftarrow 0$

        ▷ Capture case in which S is composed of one or all elements of A.
24:      **for each** x in range [1, length(A)] **do**
25:          **if** $A_x$ == c **then**
26:              Append $A_x$ to $S$
27:              **return** $S$
28:          **end if**
29:          $runningSum \leftarrow runningSum + A_x$
30:          **if** $runningSum$ == c **then**
31:              **for each** y in range [1, x] **do**
32:                  Append $A_y$ to $S$.
33:              **end for**
34:              **return** $S$
35:          **end if**
36:      **end for**

        ▷ Capture case in which S is composed of all but one element of A.
37:      **for each** x in range [1, length(A)] **do**
38:          **if** $runningSum$ - $A_x$ == c **then**
39:              **for each** y in range [1, length(A)] **do**
40:                  **if** y != x **then**
41:                      Append $A_y$ to $S$.
42:                  **end if**
43:              **end for**
44:              **return** $S$
45:          **end if**
46:      **end for**


47:      **if** length(A) <5 **then**
48:          **return** S
49:      **end if**

        ▷ Construct list of pairs existing within the set; capture case in which S is
        composed of two elements.
50:      **for each** i in range [1, length(A)] **do**
51:          **for each** j in range [i+1, length(A)] **do**

52:             **if** $A_i + A_j == c$ **then**
53:                Append $A_i$ to $S$.
54:                Append $A_j$ to $S$.
55:                **return** $S$
56:             **end if**
57:             **if** $A_i\ != A_j$ **then**
58:                Append $\{A_i, A_j\}$ to *pairs*.
59:             **end if**
60:         **end for**
61:      **end for**

    ▷ At this point, all cases for sets of length less than or equal to four have been captured. Formulate all possible window configurations of $A$ (order $n^4$ in number).
62:      **for each** i in range $[1, \text{length}(pairs)]$ **do**
63:         **for each** j in range $[i+1, \text{length}(pairs)]$ **do**
64:            **if** $\text{length}(\text{pairs}_i \textbf{ intersect } \text{pairs}_j) == 0$ **then**
65:               Append $\{\text{pairs}_{i,1}, \text{pairs}_{i,2}, \text{pairs}_{j,1}, \text{pairs}_{j,2}\}$ to *windows*.
66:            **end if**
67:         **end for**
68:      **end for**

    ▷ Apply the matrix-based strategy whose convention is given in 2.1.
69:      **for each** i in range $[1, \text{length}(windows)]$ **do**
70:         **swap** the elements of $windows_i$ into the first four positions of A.
71:         **for two iterations do**
72:            **for each** t in range$[3, \text{length}(A) - 1]$ **do**
73:               CONSTRAIN(A, S, c, t)
74:               **if** $\text{length}(S) > 0$ **then**
75:                  **return** S
76:               **end if**
77:            **end for**
78:            **swap** the second element with the third
79:         **end for**
80:      **end for**
81:      **return** $S$
82: **end procedure**

    ▷ CONSTRAIN constrains a set to the parameterized linear constraints of Z, updating the subset list if the characteristic subset membership vector can be formed.
83: **procedure** CONSTRAIN(A,S,c,t)
84:      $m \leftarrow BALANCE(A, c, t, 1, 1)$
85:      **if** $m\ != \text{null}$ **then**
86:         **for each** index x of m for which $m_x == 1$ **do**
87:            Append $A_x$ to S

```
88:         end for
89:         return
90:      end if
91:      m ← BALANCE(A, c, t, 0, 0)
92:      if m != null then
93:         for each index x of m for which m_x == 1 do
94:             Append A_x to S
95:         end for
96:         return
97:      end if
98:      m ← BALANCE(A, c, t, 1, 0)
99:      if m != null then
100:         for each index x of m for which m_x == 1 do
101:            Append A_x to S
102:         end for
103:         return
104:      end if
105:      m ← BALANCE(A, c, t, 0, 1)
106:      if m != null then
107:         for each index x of m for which m_x == 1 do
108:            Append A_x to S
109:         end for
110:         return
111:      end if
112: end procedure
```

▷ BALANCE attempts to form $m$ from the solution space of Z

```
113: procedure BALANCE(A, c, t, v_r, v_s)
114:      m ← TEST(A, c, t, 1, 1, v_r, v_s)
115:      if m is not null then
116:         return m
117:      end if
118:      m ← TEST(A, c, t, 0, 0, v_r, v_s)
119:      if m is not null then
120:         return m
121:      end if
122:      m ← TEST(A, c, t, 1, 0, v_r, v_s)
123:      if m is not null then
124:         return m
125:      end if
126:      m ← TEST(A, c, t, 0, 1, v_r, v_s)
127:      if m is not null then
128:         return m
129:      end if
130:      return null
131: end procedure
```

```
132: procedure TEST(A, c, t, t₁, t₂, vᵣ, vₛ)
133:     Initialize empty list S.
134:     Initialize empty list tmp.
135:     if t₁ equals 1 then
136:         append A₁ to S
137:     end if
138:     if t₂ equals 1 then
139:         append A₂ to S
140:     end if
141:     if vᵣ equals 1 then
142:         append A₃ to S
143:     end if
144:     if vₛ equals 1 then
145:         append A₄ to S
146:     end if
147:     testSum ← 0
148:     for each s in S do
149:         testSum ← testSum + s
150:     end for
151:     if testSum equals c then
152:         return S
153:     end if
154:     sumTop ← 0
155:     sumBottom ← 0
156:     for each x in range [1, length(A)-4] do
157:         calculate D₁,ₓ
158:         calculate D₂,ₓ
159:         calculate D₃,ₓ
160:         if 0 <D₃,ₓ <1 then
161:             append x to tmp
162:             sumTop ← sumTop + D₁,ₓ
163:             sumBottom ← sumBottom + D₂,ₓ
164:             if sumTop equals 1 and sumBottom equals 1 then
165:                 for each i in tmp do
166:                     append Aᵢ₊₄ to S
167:                 end for
168:                 return S
169:             end if
170:         end if
171:     end for
172:     return null
173: end procedure
```

## 2.4 Complexity

This algorithm satisfies the classification of strongly polynomial in that (1) the number of operations in the arithmetic model of computation is bounded in a polynomial in the number of integers in the input and (2) the space used by the algorithm is polynomial in the size of the input.[1]

(1) The TEST procedure requires one linear pass through input set $A$, requiring a constant number of arithmetic computations at each element to calculate the directional contribution table values, followed by at most n appendages to $S$. The number of arithmetic computations of TEST is O(n). The BALANCE procedure requires four repetitions of the TEST procedure: the number of arithmetic computations required by BALANCE is O(n). The CONSTRAIN procedure requires four repetitions of the BALANCE procedure, followed by at most n appendages to $S$. The number of arithmetic computations required by CONSTRAIN is O(n). The PARTIALSUBSETSUM procedure requires $n^4$ iterations of $n$ iterations of the CONSTRAIN procedure. SUBSETSUM requires at most n iterations of PARTIALSUBSETSUM. The number of arithmetic computations required by SUBSETSUM is O($n^7$).

(2) The TEST procedure in one execution requires the space taken by $A$, the space taken by $S$ which is less than or equal to the length of $A$, and a finite, constant number of integers of bit length less than or equal to two times the number of bits of the maximum element of $A$: TEST is O(n) in required space, where n is the number of bits in the input set. BALANCE requires the same space as TEST. CONSTRAIN requires the same space as BALANCE. SUBSETSUM requires storage of all possible length-4 subsets of $A$, meaning that the total space required by the algorithm is bounded by O($n^4$).

# 3 Discussion

## 3.1 Accuracy Results

A simple implementation of this algorithm was written (hosted on Github at [https://github.com/ad-alston/PolynomialSubsetSum](https://github.com/ad-alston/PolynomialSubsetSum)) in Java using 64-bit long integers and tested for accuracy to reveal efficacy of the algorithm.

To test the accuracy of the algorithm, a driver was implemented which generates random sets of integers of a given length $n$, having range $-2n^2$ to $2n^2$ and tests whether the set satisfies subset sum for $c$ equal to 0 using (a) the conventional exponential algorithm and (b) the SUBSETSUM routine for $n$ permutations of the set. Under the parameters of this test, failure occurs when the output of (b) differs from that of (a).

For each $n$ from 1 to 20, $1,000,000$ such sets were generated and used to test the implementation of the algorithm. Following all $20,000,000$ trials, the success rate was 100%, exhibiting precisely 0 failures.

---

[1]Definition of strongly polynomial algorithm: Grotschel, Martin; Laszlo Lovasz, Alexander Schrijver (1988). "Complexity, Oracles, and Numerical Computation". Geometric Algorithms and Combinatorial Optimization. Springer.

To further explore general-case applicability of the algorithm, a reduction to subset sum from 3-SAT, another NP-complete problem, was implemented and tested, also exhibiting precisely 0 failures over all trials.

## 3.2   Reduction to an Approximation

The method can be reduced to an $O(n^4)$ approximation method by only using the subset of possible window configurations represented by shifting the entire set to the left (wrapping the element of the first index to the last position in the set) n times and repeating the CONSTRAIN procedure. Performing the same testing as was performed on the exact method as was outlined in section 2.2, yielding a success rate of 99.95% over $20,000,000$ trials.

The fact that such accuracy is yielded from a derived approximation which performs in the somewhat practical time of $O(n^4)$ gives reason to believe that such a method may call into question the true reliability of SSP- and NP-complete-reliant cryptosystems, given that this complexity is not dependent on the range or values in the input set.

# 4   Conclusion

An algorithm has been provided which reduces the conditions under which a given set satisfies the stipulations of the subset sum proposition to a set of linear relationships, answering question of whether a set satisfies subset sum may be answered in a number of steps strongly polynomial with respect to the length of the input. Following justification, implementation, exploration of applications, and testing of this algorithm, a rate of accuracy and observed applicability was yielded which calls to question the reliability of cryptosystems resting on the assumption that large instances of NP-complete problems cannot be solved in a practical amount of time by conventional computers.