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. Following implementation of this algorithm, initial testing over $20,000,000$ trials using random inputs of length 1 to 20, $1,000,000$ trials each, reveal 100% accuracy with exactly 0 failures.

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-N))*, where n is the length of the set and *B-A* is the range of inputs; however, this solution is not truly polynomial, as it is polynomial with respect to *M-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.

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 or equal to 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 ith value in m is 1 if the ith 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.

$$a_1m_1 + a_2m_2 + ... + a_nm_n = c \tag{3}$$

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 rth 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 sth 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:

$$\begin{aligned} Z(A, t, r, v_r, s, v_s):& \\ & a_1m_1 + a_2m_2 + ... + a_nm_n = c \\ & m_1 + m_2 + ... + m_n = t \\ & m_r = v_r \\ & m_s = v_s \end{aligned} \tag{7}$$

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'$_5$, . . . 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, . . ., n* to be 1, all other m_j to be 0. This convention then allows *m'* for any *A'* to be expressed as follows:

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

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 $\mathrm{v_r}$ and $\mathrm{v_s}$) 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) properties of have been determined and are employed by the algorithm to determine *m'* encoding *S* within *A'*:

(A) If b_1 or b_2 (but not both) is already $\mathrm{t_1}$ or $\mathrm{t_2}$ respectively, *m'* may be formed by taking the vectors represented by the columns *i* of *D* corresponding to positive values in row 2 of *D* (if b_1) or row 1 of *D* (if b_2).

(B) If there exist columns for which $D_{3,i}$ have equivalent absolute values, *m'* may be formed by taking the vectors represented by each column *i*.

(C) If there exist values in row 3 of *D* having absolute value greater than one, *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.

(D) *m'* may be formed by taking the null space vectors represented by the columns *i* corresponding to the $t - v_r - v_s - t_1 - t_2$ least absolute values in the third row of *D*.

2.2 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.

```
procedure SubsetSum($A, c$)
    Initialize empty list $S$.
    Initialize empty list $pairs$.
    Initialize empty list $windows$.
    $runningSum \leftarrow 0$

  ▷ Capture case in which S is composed of one or all elements of A.
    for each x in range [1, length(A)] do
        if $A_x$ == c then
            Append $A_x$ to $S$
            return $S$
        end if
        $runningSum \leftarrow runningSum + A_x$
        if $runningSum$ == c then
            for each y in range [1, x] do
                Append $A_y$ to $S$.
            end for
            return $S$
        end if
    end for

  ▷ Capture case in which S is composed of all but one element of A.
    for each x in range [1, length(A)] do
        if $runningSum$ - $A_x$ == c then
            for each y in range [1, length(A)] do
                if y != x then
                    Append $A_y$ to $S$.
                end if
            end for
            return $S$
        end if
    end for


    if length(A) <5 then
        return S
    end if

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

```
            if $A_i + A_j == c$ then
                Append $A_i$ to $S$.
                Append $A_j$ to $S$.
                return $S$
            end if
            Append $\{A_i, A_j\}$ to $pairs$.
        end for
    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).
    for each i in range [1, length($pairs$)] do
        for each j in range [i+1, length($pairs$)] do
            if length($pairs_i$ intersect $pairs_j$) == 0 then
                Append $\{pairs_{i,1}, pairs_{i,2}, pairs_{j,1}, pairs_{j,2}\}$ to $windows$.
            end if
        end for
    end for

▷ Apply the matrix-based strategy whose convention is given in 2.1.
    for each i in range [1, length($windows$)] do
        swap the elements of $windows_i$ into the first four positions of A.
        for two iterations do
            for each t in range[3, length(A) - 1] do
                CONSTRAIN(A, S, t, 0, 0)
                if length(S) >0 then
                    return S
                end if
                CONSTRAIN(A, S, t, 1, 1)
                if length(S) >0 then
                    return S
                end if
                CONSTRAIN(A, S, t, 0, 1)
                if length(S) >0 then
                    return S
                end if
                CONSTRAIN(A, S, t, 1, 0)
                if length(S) >0 then
                    return S
                end if
            end for
            shift the four-element window to the left, wrapping element 1 to index 4.
        end for
    end for
    **return** $S$
**end procedure**

$\triangleright$ CONSTRAIN constrains a set to the parameterized linear constraints of
Z, updating the subset list if the characteristic subset membership vector
can be formed.
**procedure** CONSTRAIN(A, S, t, $v_r$, $v_s$)
    **form** Z(A, t, 3, $v_r$, 4, $v_s$)
    **solve** Z to obtain S(Z)
    $m \leftarrow BALANCE(S(Z))$
    **if** $m$ != null **then**
        **for each** index x of m for which $m_x == 1$ **do**
            Append $A_x$ to S
        **end for**
    **end if**
**end procedure**

$\triangleright$ BALANCE attempts to form $m$ from the solution space of Z
**procedure** BALANCE(S(Z))
    **form** D(S(Z), 0, 0)
    **if** property (A), (B), (C), or (D) applies to D **then**
        **for each** applicable vector v **do**
            **add** v to the particular solution of S(Z)
        **end for**
        **return** particular solution of S(Z)
    **end if**
    **form** D(S(Z), 1, 1)
    **if** property (A), (B), (C), or (D) applies to D **then**
        **for each** applicable vector v **do**
            **add** v to the particular solution of S(Z)
        **end for**
        **return** particular solution of S(Z)
    **end if**
    **form** D(S(Z), 0, 1)
    **if** property (A), (B), (C), or (D) applies to D **then**
        **for each** applicable vector v **do**
            **add** v to the particular solution of S(Z)
        **end for**
        **return** particular solution of S(Z)
    **end if**
    **form** D(S(Z), 1, 0)
    **if** property (A), (B), (C), or (D) applies to D **then**
        **for each** applicable vector v **do**
            **add** v to the particular solution of S(Z)
        **end for**
        **return** particular solution of S(Z)
        end if
        return null
end procedure
```

2.3 Complexity

The complexity of the given algorithm is polynomial with respect to n, the length of the input set. The set of checks present for an input of length less than or equal to four are of complexity O(n^2). For sets of length greater than four, formation of the window configurations of A is completed in n choose four steps, O(n^4) time. The BALANCE procedure can be performed in O(n^2) time using a set of sixteen pairwise searches through D, followed by n additions for an order of n possible applicable vectors. The CONSTRAIN procedure can be performed in O(n^2) time by performing simple forward elimination and back-substitution on a 4xn matrix, followed by the BALANCE procedure which itself is O(n^2) in terms of complexity. The SUBSETSUM procedure, then, performs in n^4 iterations of n times n^2 steps, giving the algorithm a total complexity of O(n^7).

3 Discussion

3.1 Accuracy Results

A simple implementation of this algorithm was written (hosted on Github at `https://github.com/ad-alston/PolynomialSubsetSum`) in Java using floating-point precision numbers (implementation is possible using arbitrary precision sets of two integers to represent rational numbers) 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.

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.

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 polynomial number of steps. Following justification, implementation, and testing of this algorithm, $20,000,000$ trials using random inputs of length 1 to 20, $1,000,000$ trials each, reveal 100% accuracy with exactly 0 failures, in alignment with general-case applicability of this algorithm.