Solving subset sum in polynomial time and prooving P = NP

Samir Bouftass

E-mail : crypticator@gmail.com

November 18, 2024

Abstract

In this paper, we show that subset sum problem is solvable in polynomial time.

Keywords : Subset sum problem, Polynomial time, NP complete.

1 Introduction :

Subset sum is a famous problem in computer science, shown to be NP complete [1], it consists on deciding whether there is a substet of integers belonging to a set that sums to a given target sum integer. In this paper we show that varient of subset sum in which all inputs are positive could be solved in polynomial time, this varient is also NP complete [1][2].

2 (b,d) Vectors :

Definition 2.1. *A (b, d) vector corresponding to number n is a vector V satisfying the following equality : $n = V_0 b^0 + \ldots.. + V_{d-1} b^{d-1}$, where V_i $(0 \leq i < d)$ components of V are positive numbers.*

Definition 2.2. *Let (b, d) vector V $[V_0 ... V_i \ V_{(i+1)} \ldots V_{d-1}]$ corresponding to number n.*

Carry up ith component of V is defined by operations below
$V_i = V_i - b$.
$V_{(i+1)} = V_{(i+1)} + 1$.
Constrained Carry up requires $V_i > b$.

Carry down ith component of V is defined by operations below :
$V_{(i+1)} = V_{(i+1)} - 1$.
$V_i = V_i + b$.
Constrained Carry down requires $V_{(i+1)} > 1$.

Definition 2.3. *Let two vectors V1 and V2 . abs distance between V1 and V2 is* $\sum_i |V1_i – V2_i|$.

Definition 2.4. *abs modulus of vector V is* $\sum_i |V_i|$.

Proposition 2.1. ***There is at least one*** (b, d) ***vector corresponding to a number*** n***.***

Proof.
Let $[V_0\ V_1\ \ldots\ V_{d-1}]$ *be a* (b, d) *vector that corresponds to a number* n*.*
$n = V_0b^0 + V_1b^1 + \ldots.. + V_{d-1}b^{d-1}$*, where* $V_1 > 1$*.*
By constrained carry down V_0 *, we get :*
$n = (V_0 + b)b^0 + (V_1 - 1)b^1 + \ldots.. + V_{d-1}b^{d-1}$*. Meaning* $[V_0 + b\ V_1 - 1\ \ldots\ V_{d-1}]$ *is also a* (b, d) *vector corresponding to* n*.* □

Proposition 2.2. ***It is easy to find a*** (b, d) ***vector corresponding to a number*** n***.***

Proof.
We will proceed by proving by construction
We represent n *in base* b*.*
We automatically get a (b,s) vector V corresponding to n, s being n size in base b.
if $s < d$*, we extend V size to d by filling its components of which indexes are greater than s, by zeroes.*
if $s < d$*, we replace V dth component value by* $n/2^{d-1}$*.*

Proposition 2.3. ***Let a (b,d) vector V, constrained carry down its components increases resulting (b,d) vector Abs modulus. Conversely constrained carry up decreases it.***

Proof.
Note, If we constrained carry up V_i *, abs modulus decreases by :*
$b - 1 = |V_i - b| + |V_{(i+1)} + 1|$.
If we constrained carry down V_i*, abs modulus increases by :*
$b - 1 = |V_i + b| + |V_{(i+1)} - 1|$.

Lemma 2.1. ***Let a (b,d) vector V1, there is only one (b,d) vector V2 corresponding to a number n that minimizes Abs distance between V1 and V2.***

Proof.
Let m be the number that V1 corresponds to.To find $V2$*, we choose a* (b,d) *vector* $V3$ *corresponding to* $|m-n|$ *such as* $V3_i < b$ *for* $i < d-1$*.*
Then compute $V2 = V1–V3$ *if* $n \leq m$*,* $V2 = V1 + V3$ *otherwise.*
And carry down negative V2 components to fulfill (b,d) vector condition :
$0 < V2_i$ *for* $0 \leq i < d$*.*
Observe, V3, (b,d) vector corresponding to $|m-n|$ *is the closest one can get to nil vector* ***(proposition 3)*** *. Indeed, because all components of V3 are inferior to b where* $i < d$*, we can't carry up nor carry down to decrease the abs distance.*

□

Theorem 2.1. ***Let V1 and V2 be 2 different (b,d) vectors corresponding to the same number n. There is a polynomial time algorithm that transforms V2 to V1.***

Proof.
We will proceed by proving by construction, pseudo code below transforms V2 to V1.
Observe, this pseudo code transforms V2 components one by one, its complexity is O(n²).

Algorithm 1 Pseudo code 1

```
i ← 0
while i < d do
    if V2_i > V1_i then
        repeat Carry up V2_i
        until V2_i = V1_i
    end if
    if V2_i < V1_i then
        repeat Carry down V2_i
        until V2_i = V1_i
    end if
    i ← i + 1
end while
```

□

The following pseudo code complexity is also in $O(n^2)$. it uses abs distance to adjust all the components of V2 in one loop, whereas in the former, components of V2 are adjusted sequentially.

Algorithm 2 Pseudo code2. Input : V1, V2. Output : V2

```
i ← 0
dist ← abs_dist(V1, V2)
updated_dist ← 0
while i < d do
    V3 ← V2
    constrained carry up  V3_i
    updated_dist ← abs_dist(V3, V1)
    if updated_dist < dist  then
        V2 ← V3
        dist ← updated_dist
    end if
    if dist < updated_dist  then
        V3 ← V2
        constrained carry down  V3_i
        updated_dist ← abs_dist(V3, V1)
        if updated_dist < dist  then
            V2 ← V3
            dist ← updated_dist
        end if
    end if
    i ← i + 1
    if dist = 0  then return V2
    end if
    if i = d  then
        i ← 0
    end if
end while
```

Theorem 2.2. ***Let V1 and V2 be 2 different (b,d) vectors, V2 corresponds to number n. There is a polynomial time algorithm that finds V3 the closest (b,d) vector to V1, corresponding to n.***

Proof.
According to ***Lemma 2.1****, we know that V3 exists. We adapt pseudo code 2 to find V3.*
Observe, in a loop all possible carry ups and carry downs of components of V3 are performed, meaning $|V1–V2|$ *is minimized if* $abs\_distance(V1, V2)$ *don't decrease after a loop.*

Algorithm 3 Pseudo code 3. Input V1, V2. Output : V3

```
i ← 0
dist ← abs_dist(V1, V2)
updated_dist ← 0
dist1 ← 0
while i < d do
    V3 ← V2
    constrained carry up  V3_i
    updated_dist ← abs_dist(V3, V1)
    if updated_dist < dist  then
        V2 ← V3
        dist ← updated_dist
    end if
    if dist < updated_dist  then
        V3 ← V2
        constrained carry down  V3_i
        updated_dist ← abs_dist(V3, V1)
        if updated_dist < dist  then
            V2 ← V3
            dist ← updated_dist
        end if
    end if
    i ← i + 1
    if i = d and dist ≠ 0  then
        if  dist = dist1 then return V3
        end if
        dist1 ← dist
        i ← 0
    end if
end while
```

□

3 Subset sum complexity class :

Addition seen differently :

Addition of 2 numbers can be performed by first adding their corresponding (b, d) vectors. b is the base where they are represented . Carry propagation is realized by extending sum vector V size and carry up its components until their values become inferior to b.

Definition 3.1 . *Given a set S of (b,d) vectors and and a (b,d) target vector T. Subset sum without carrying problem, consists on finding a subset of vectors Sb that sums to T.*

Definition 3.2 . *Let a vector V, abs distance to binary of V is* $\sum_i ||V_i – 0.5| - 0.5|$. *abs distance to binary of V equals 0, imply components of V are in* $\mathbb{N}_2$.

Proposition 3.1 . ***Subset sum without carrying is in P.***

Proof.
Observe, solving subset sum without carry is equivalent to find solutions of following linear equation $\boldsymbol{A}X = T$ *where components of a column of matrix* $\boldsymbol{A}$ *are components of a vector in set S.*
If subset Sb exists, components of solution X are in $\mathbb{N}_2$.
If $X_i = 0$, S_i *ith vector in S, is not in subset Sb.*
If $X_i = 1$, S_i *ith vector in S, is in subset Sb.*

Observe, Hardness of Subset sum resides mainly in carry propagation complexity, that's sort of hiding the target vector and showing a number it corresponds to. To Solve subset sum efficiently, man had to find the right (b,d) vector T corresponding to target t such as $X = \boldsymbol{A}^{-1}T$ *components are in* $\mathbb{N}_2$ *meaning :* $\sum_i ||(\boldsymbol{A}^{-1}T)_i – 0.5| - 0.5| = 0$.
NB : If matrix $\boldsymbol{A}$ *is not invertible we use gaussian elimination to solve* $\boldsymbol{A}X = T$. □

Proposition 3.2 . ***Let V :*** $[V_0\ V_1\ \ldots\ V_{d-1}]$ ***be a*** (b,d) ***vector corresponding to a number*** n***, and a number*** $u < V_{d-1}$***. (b,d) vector :*** $[(V_0 + 2 \times u)\ (V_1 + u)\ \ldots\ (V_{d-1} - u)]$ ***corresponds to*** n.

Proof.
Observe u carry down V (d-1)th component gives also a (b,d) vector corresponding to n which is : $[V_0\ V_1\ \ldots\ (V_{d-2} + 2 \times u)\ (V_{d-1} - u)]$. If we u right permute V ith component to its (i-1)th component from $i = d - 2$, we get also a (b,d) vector corresponding to n which is : $[V_0 + u \times 2\ V_1 + u\ \ldots\ V_{d-1} - u]$.

□

Theorem 3.1. ***Subset set sum is in P.***

Proof.
Let S be a set of numbers whose maximal size in bits is d and a target t.
Observe the binary representations of S elements are (2,d) vectors that corresponds to them.
Ts is a (2,s) vector corresponding to t where s is t size in bit. By ***proposition 3.2****, it is easy to show that* $T = [(Ts_0 + 2 \times (t/2^d))\ (Ts_1 + (t/2^d))\ \ldots\ (Ts_{d-1} + (t/2^d))]$ *is a (b,d) vector that corresponds to t.*
$\boldsymbol{A}$ *is a matrix which columns are (2,d) vectors corresponding to elements of S.*

To find if a subset of S sums to t, we execute pseudo code 4 which is basically the same as pseudo code 3, it transforms T to a (2,d) vector that is closest to vectors over $\mathbb{N}_2$ *in basis A.*
Because solving linear system of equation complexity is $O(n^3)$*, according to* ***theorems 2.1 & 2.2*** *Pseudo code 4 complexity is* $O(n^5)$*. If final computed distant is nil, transformed T components in base A (X components) are in* $\mathbb{N}_2$*, meaning there is a subset of S that sums to t, otherwise there is no subset of S that sums to t.*

Algorithm 4 Pseudo code 4. Inputs S : A, T. Outputs : T, X

```
Solve AX = T
dist ← abs_dist2binary(X)
updated_dist ← 0
dist1 ← 0
while i < d do
    T1 ← T
    constrained carry up T1_i
    Solve AX = T1
    updated_dist ← abs_dist2binary(X)
    if updated_dist < dist then
        T ← T1
        dist ← updated_dist
    end if
    if dist < updated_dist then
        T1 ← T
        constrained carry down T1_i
        Solve AX = T1
        updated_dist ← abs_dist2binary(X)
        if updated_dist < dist then
            T ← T1
            dist ← updated_dist
        end if
    end if
    i ← i + 1
    if i = d and dist ≠ 0 then
        if dist = dist1 then return T,X
        end if
        dist1 ← dist
        i ← 0
    end if
end while
```

□

Corollary 3.1. ***P = NP.***

Proof.
Subset sum is in an NP complete problem meaning that if it is in P all the problems in NP are in it too [1][2]. Proof follows directly from ***theorem 3.1****.* □

4 Conclusion :

Subset sum may be considered as equivalent to a two stage algorithm.

In the first stage :

Matrix **A** is a given, vector X over $\mathbb{N}_2$ is unknown . The first stage output is vector $T = \mathbf{A}X$.

In the second stage :

Vector T which is also a (b,d) vector for some b and d, is transformed to integer n it corresponds to. $n = T \bullet U$ where U is vector $[(2^0)(2^1)....(2^{d-2})(2^{d-1})]$

Solving subset sum consists on finding a solution over $\mathbb{N}_2$ of equation $n = \mathbf{A}X \bullet U$ where U = $[(2^0)(2^1)....(2^{d-2})(2^{d-1})]$

In this paper, we showed that it is easy to find a (b,d) vector corresponding to an integer n.

We equally showed transforming a (b,d) vetor corresponding to integer n to another vector, it corresponds to can be performed in polynomial time by decreasing distance between them to zero. **(theorem 2.1 and 2.2)**.

In subset sum we dont know the final (2,d) vector V, we had to transform to, but we know that its components are in $\mathbb{N}_2$: they verify $||V_i - 0.5| - 0.5| = 0$. To capture this proprety we introduced abs 2 binary distance, and showed that the "first stage" is also easy to invert. **(theorem 3.1)**.

References

[1] Stephen A. Cook *The complexity of theorem-proving procedures, STOC (1971) : Proceedings of the third annual ACM symposium on Theory of computing Pages 151 - 158*

[2] Richard M Karp. *Reducibility Among Combinatorial Problems*, (1972), Complexity of Computer Computations, Springer Verlag , Berlin Heidelberg .