I.J. Mathematical Sciences and Computing, **2017, 2, 55-61**
Published Online April 2017 in MECS (http://www.mecs-press.net)
DOI: 10.5815/ijmsc.2017.02.05

Available online at ***http://www.mecs-press.net/ijmsc***

A Fast Heuristic Algorithm for Solving High-Density Subset-Sum Problems

Akash Nag

Dept. of Computer Science, The University of Burdwan, Rajbati, Burdwan 713104, India

Abstract

The subset sum problem is to decide whether for a given set of integers *A* and an integer *S*, a possible subset of *A* exists such that the sum of its elements is equal to *S*. The problem of determining whether such a subset exists is NP-complete; which is the basis for cryptosystems of knapsack type. In this paper a fast heuristic algorithm is proposed for solving subset sum problems in pseudo-polynomial time. Extensive computational evidence suggests that the algorithm almost always finds a solution to the problem when one exists. The runtime performance of the algorithm is also analyzed.

Index Terms: Subset-sum problem, NP-complete, heuristics, search, algorithms.

1. Introduction

The *subset sum problem* is a NP-complete problem [1] and is defined as follows: Given a set $A = \{a_i : 1 \le i \le n\}$ of integers (usually called *weights*) and an integer *S* (called the *sum*), determine whether or not there exists a subset *B* of *A*, such that:

$$\sum_{i=1}^{k} b_i = S \qquad \forall b_i \in B, k = |B|, B \subseteq A \tag{1.1}$$

Besides the decision problem, an exact solution to the problem would be to additionally determine *B* (given *A* and *S*), provided such a *B* exists. This latter problem can be alternatively defined in the form of a 0-1 *knapsack* or 0-1 integer programming problem: determine a vector *x* of size *n*, such that:

* Corresponding author.
E-mail address:

$$\sum_{i=1}^{n} a_i.x_i = S \qquad \forall a_i \in A, x_i \in \{0,1\}, n = |A| \tag{1.2}$$

Several public-key cryptosystems are based on this problem, the most basic such system being the Merkle-Hellman cryptosystem [2], which was broken by Shamir in 1983 [3]. Such systems use a public-key, which is the given set of weights A, and the binary representation of the plaintext is the vector *x* in Eq. 1.2. The calculated sum *S* is then transmitted to the receiver as the cipher-text.

Decrypting the cipher-text *S* is then an instance of the subset-sum problem, which is known to be NP-hard and hence difficult to solve. However, the weights are chosen in such a way so that the problem is easily solved by the received given knowledge of some *trapdoor* information.

The density *D* of the set of weights $A = \{a_1, a_2, ..., a_n\}$, is then defined as:

$$D = \frac{n}{\log_2 \max_{1 \le i \le n}(a_i)} \tag{1.3}$$

For the purposes of this paper, the bounds on the weights is defined to be the range *R* of the set, i.e. $\left(1 - R/2\right) \le a_i \le \left(R/2 - 1\right) \forall 1 \le i \le n$. A novel algorithm, called the *CP Algorithm* (*Card-Players' Algorithm*), is now proposed for solving the subset-sum problem.

In Section 2, we look at some relevant work that has been done in this field, and in Section 3, we present the proposed algorithm using the analogy of a card-game. In Section 4, we present our results and discuss its performance, and conclude with remarks about the current state of this field in Section 5.

2. Related Work

The hardness of solving the subset-sum problem varies directly with the density of the set of weights. There can be two cases:

a) **D≤1:** These low-density subset-sum problems are efficiently solved by reduction to a short vector in a lattice, as presented by Brickell [4]; Lagarias and Odlyzko [5]; Martello and Toth [6]; Coster et al. [7].

b) **D>1:** These medium and high-density subset-sum problems are solvable by dynamic programming techniques or using analytical number theory, such as those presented in Chaimovich et al. [8]; Galil and Margalit [9]; Flaxman and Przydatek [10]; with some of these failing to find a solution if certain bounds for *n* or *R* are not respected.

In this paper, a novel, heuristic but non-deterministic, algorithm is proposed for the latter case of high-density problems.

3. The Proposed Algorithm

3.1. Overview

The proposed algorithm is a non-deterministic iterative algorithm, and is hereafter called the *Card-Players' Algorithm* (*CP Algorithm*), after the manner in which the solution is obtained. The algorithm is best explained using an analogy of two card players: Alice and Bob, playing a game of cards. The deck of cards with which

the game is played is not the usual 52-card deck, but is rather an *n*-card deck with each card numbered with an integer *a* in the range [1-(*R*/2), (*R*/2)-1]. The input to the algorithm, which is the set of integers *A*, is therefore analogous to the deck of cards in this game.

The game is a challenge game, where Bob picks a number *S* (or the *sum*, being the other input to the algorithm) at the beginning of the game, and challenges Alice to come up with a hand in which the cards add up to *S*.

3.2. Initialization

The game begins by shuffling the deck, and then dealing it equally among Alice and Bob, whereupon each end up with *n*/2 cards in hand. If *n* is odd, Alice will end up having one more card than Bob. Bob then chooses a random number S_{OBJ}, and challenges Alice to come up with a hand in which the cards add up to S_{OBJ}.

Formally, let H_{Alice} and H_{Bob} be the two hands of Alice and Bob, *A* be the total deck of cards, and *n* be the total number of cards in the deck. At the beginning of the game, these are defined as:

$$\begin{aligned} &A = \{a_i : 1 \le i \le n\} && \forall a_i \in [1 - R/2, R/2 - 1] \\ &H_{Alice} = \left\{ x_i : 1 \le i \le \left(\left\lfloor \frac{n}{2} \right\rfloor + n \bmod 2 \right) \right\} && \forall x_i \in A \\ &H_{Bob} = \left\{ y_i : 1 \le i \le \left\lfloor \frac{n}{2} \right\rfloor \right\} && \forall y_i \in A, H_{Alice} \cap H_{Bob} = \phi \end{aligned} \tag{3.1}$$

3.3. Objective

The objective of the game, for Alice, is to win the challenge thrown by Bob, i.e. to come up with a sum of S_{OBJ}. During the entire duration of the game, Bob only plays to help Alice achieve her goal, and has no separate objective of his own.

3.4. Game Play

Prior to the beginning of *each* turn, an ordered pair formed from the sum of Alice's hand and that of Bob's hand are recorded. Whenever, *after* a turn, such a sum-pair is repeated in the game, a fair coin is tossed, and depending on the outcome – either Alice or Bob randomly chooses a card from her/his hand and gives it to the other. After such a transfer, the sum-pair is again checked for repetition and if it repeats, the coin toss is repeated again until either the number of such consecutive tosses equals *n*, or a unique sum-pair is obtained. The game proceeds if a unique sum-pair is obtained; otherwise, Alice loses the game and declares that she cannot win the challenge.

Let S_{Alice} and S_{Bob} define the sum of all the cards in Alice's and Bob's hand respectively. Then the sum-pair (S_{Alice}, S_{Bob}) must be unique at every turn of the game. These are defined as follows:

$$\begin{aligned} S_{Alice} &= \sum_{i=1}^{n(H_{Alice})} x_i \qquad \forall x_i \in H_{Alice} \\ S_{Bob} &= \sum_{i=1}^{n(H_{Bob})} y_i \qquad \forall y_i \in H_{Bob} \end{aligned} \tag{3.2}$$

At each turn, Alice makes her decision based on two values: her requirement (denoted by *REQ*), and the Closest-Element (denoted by *CE*). The former is calculated by Alice herself, while she asks Bob for the latter. These are defined as:

$$\begin{aligned} &REQ = S_{OBJ} - S_{Alice} \\ &CE = \begin{Bmatrix} y_k & f(REQ) \neq \phi, d_k = \min(d_i) \forall (y_i, d_i) \in f(REQ), y_i \in H_{Bob} \\ \phi & f(REQ) = \phi \end{Bmatrix} \\ &f(p) = \left\{(y,d) : SIGN(y,p) = 1, |y| \leq |p|, d = \big||y| - |p|\big|\right\} \\ &SIGN(x,y) = \begin{Bmatrix} 1 & x < 0, y < 0 \\ 1 & x \geq 0, y \geq 0 \\ 0 & otherwise \end{Bmatrix} \end{aligned} \tag{3.3}$$

Informally, Alice computes what card she needs to achieve her objective, and then asks Bob if he has that card (i.e. *CE*). Bob then checks his hand to see if he has that card. If yes (i.e. $CE \neq \phi$), he gives that card to Alice. If he does not have such a card (i.e. $CE = \phi$), Alice checks how many cards she has in her hand. If she has only 1 card left, she asks Bob for a random card from his hand. If she has more than 1 card left, she calculates the *Appropriate Discardable Element* (*ADE*) as follows:

$$\begin{aligned} &G = \left\{(s_i, x_i) : s_i = S_{Alice} - x_i \ \forall x_i \in H_{Alice}\right\} \\ &ADE = \begin{Bmatrix} x_k & s_k = \min(s_i) \forall (s_i, x_i) \in G \\ \phi & G = \phi \end{Bmatrix} \end{aligned} \tag{3.4}$$

If no such element is present (i.e. $ADE = \phi$), Alice loses the game, and declares that she cannot win the challenge. If an ADE does exist however, she gives that card to Bob instead, thus completing the current turn. At the end of each turn, the winning condition is checked – if $S_{Alice} = S_{OBJ}$, then Alice wins the challenge and the solution to the subset-sum problem is Alice's hand H_{Alice}. If not, the game continues. The algorithm is presented below.

ALGORITHM 1. ***The Card-Players' Algorithm***

```
ALGORITHM CP(a[0..n-1], target)
begin
    sumList=new HashSet();
    t=(n/2)+(n mod 2);
    x[ ]=a[0..t-1];
    y[ ]=a[t..n-1];
    sx=SUM(x), sy=SUM(y);

    while (sx ≠ target) do
    begin
        rc=0;
        while (SEARCH(sumList, sx, sy) ≠ -1) do
        begin
            rc=rc+1;
            randomly transfer 1 element from x to y or vice-versa and update sx, sy accordingly
            if (rc=n) then return FAILURE;
```

```
        end
        k=GetRequirement(x, target);                    // see (3.3)
        cei=GetClosestElementIndex(y, k);               // see (3.3)
        if (cei>=0) then
            transfer y[cei] to x and update sx, sy
        else
            if (LENGTH(x)>1) then
                di=GetAppropriateDiscardIndex(x);       // see (3.4)
                if (di=-1) then return FAILURE;
                transfer x[di] to y and update sx, sy
            else
                randomly transfer 1 element from y to x and update sx, sy
            end if
        end if
        add (sx, sy) to sumList;
    end
    return x;
end
```

4. Results and Discussion

The algorithm was implemented using Java SE 8 (Update 5) on an Intel Pentium Dual-Core 2.3GHz processor, and the observed run-times and outer-loop run-counts are tabulated in Table 1, and the corresponding plot shown in Fig. 1. For *each* (*n*, *R*) pair, the experiment was repeated for 1000 random instances, and the average values for these instances (when the algorithm was successful in finding a solution) is reported in the table. The author believes that much faster run-times for this algorithm are possible if the algorithm be implemented in C/C++ and additional optimizations are applied. The outer-loop run-count is indeterminate and is dependent on both *n* and *R*, while the combined inner processes have time complexity $O(n)$, hence the time complexity of the *CP Algorithm* is $O(c.n)$ for some *c* dependent on *n* and *R*. It is evident from the table that the algorithm performs exceptionally well with regards to high density sets. The large values for *n* and the non-deterministic nature of the algorithm, made it impossible to perform an exhaustive search (using the power-set) to check indeed that no solution exists when the algorithm reported failure. But considering the fact that only very seldom were failures reported, it is strongly believed that the algorithm almost always succeeds.

Table 1. Average Run-Times (Milliseconds) and Outer-Loop Run Counts*

n	**Run-times (milliseconds)**			**Outer-loop run-counts**		
	R					
	100	**1,000**	**10,000**	**100**	**1,000**	**10,000**
50	0.000	0.078	1.062	6.347	38.393	396.106
100	0.000	0.078	1.326	5.713	21.122	204.929
200	0.015	0.078	1.249	6.010	13.296	96.133
500	0.062	0.172	1.484	8.457	11.286	41.506
1,000	0.140	0.374	2.466	12.250	12.222	27.421
2,000	0.514	0.514	4.632	17.692	15.515	22.653
5,000	2.139	1.621	11.499	32.099	23.741	26.841
10,000	7.971	4.620	21.200	57.264	33.753	34.755

** Averaged over 1000 instances, each instance consisting of a set of n random integers in the interval [1-(R/2), (R/2)-1]*

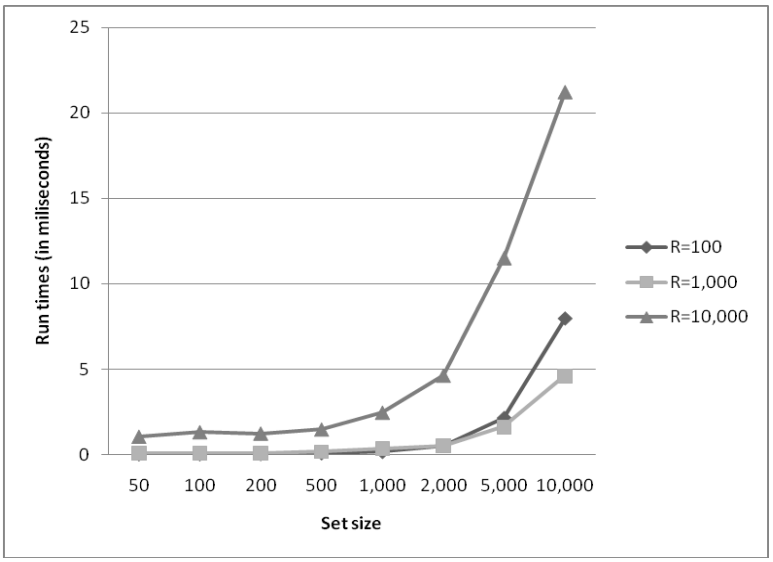

Fig.1. Plot of Execution Time vs. Set Size

5. Conclusion

The subset-sum problem is one of the most common NP-complete problems that have seen widespread research from across the computing community. Due to their applications in knapsack cryptosystems, the lion's share of the focus has been on problems with density less than 1, as discussed in Section 2, in addition to work by Daxing & Shaohan [11]. Since the compromise of the Merkle-Hellman cryptosystem by Shamir, almost all knapsack based cryptosystems have been broken, and the author believes that such public-key cryptosystems no longer have any viable future in the current form. On the other hand, high density subset sums may still have other application areas like computer passwords [12], message verification [13], RFID security [14], dealing with junk mail [15], etc., and the proposed algorithm is a positive direction in solving such problems.

References

[1] Michael R. Garey, and David S. Johnson. Computers and Intractability: A Guide to the theory of NP-Completeness. WH Freeman & Co., New York. pp:223. 1979.
[2] Ralph Merkle, and Martin E. Hellman. Hiding information and signatures in trapdoor knapsacks. Information Theory, IEEE Transactions. 24.5. pp:525-530. 1978.
[3] Adi Shamir. A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem. Advances in Cryptology. Springer, US. pp:279-288. 1983.
[4] E. F. Brickell. Solving low-density knapsacks. Advances in Cryptology, Proceedings of Crypto '83. Plenum Press, New York. pp:25-37. 1984.
[5] J. C. Lagarias, A. M. Odlyzko. Solving low-density subset sum problems. Journal of the ACM. 32(1). pp:229-246. 1985.
[6] S. Martello, and P. Toth. A new algorithm for the 0-1 knapsack problem. Management Science 34. pp:633-644. 1988.
[7] Matthijs J. Coster et al. Improved low-density subset sum algorithms. Computational complexity 2.2. pp:111-128. 1992.

[8] Mark Chaimovich, Gregory Freiman, and Zvi Galil. Solving dense subset-sum problems by using analytical number theory. Journal of Complexity 5.3. pp:271-282. 1989.
[9] Zvi Galil, and Oded Margalit. An almost linear-time algorithm for the dense subset-sum problem. SIAM Journal on Computing 20.6. pp:1157-1189. 1991.
[10] Abraham D. Flaxman, and Bartosz Przydatek. Solving medium-density subset sum problems in expected polynomial time. STACS 2005. Springer Berlin Heidelberg. pp:305-314. 2005.
[11] Daxing L., Shaohan M. (1994) Two notes on low-density subset sum algorithm. In: Du DZ., Zhang XS. (eds) Algorithms and Computation. ISAAC 1994. Lecture Notes in Computer Science, vol 834. Springer, Berlin, Heidelberg.
[12] Martello, Silvano, and Paolo Toth. "Algorithms for knapsack problems." North-Holland Mathematics Studies 132 (1987): 213-257.
[13] Sharma, Sonal, Prashant Sharma, and Ravi Shankar Dhakar. "RSA algorithm using modified subset sum cryptosystem." Computer and Communication Technology (ICCCT), 2011 2nd International Conference on. IEEE, 2011.
[14] Kate, Aniket, and Ian Goldberg. "Generalizing cryptosystems based on the subset sum problem." International Journal of Information Security 10.3 (2011): 189-199.
[15] Dwork C., Naor M. (1993) Pricing via Processing or Combatting Junk Mail. In: Brickell E.F. (eds) Advances in Cryptology — CRYPTO' 92. CRYPTO 1992. Lecture Notes in Computer Science, vol 740. Springer, Berlin, Heidelberg.

Author Profile

Mr. Akash Nag completed his Bachelors in Computer Applications from the University of Burdwan, and his Masters in Computer Science from the University of Calcutta. He is currently pursuing his Ph.D. from the Dept. of Computer Science at the University of Burdwan. He is also a guest faculty in the Dept. of Computer Science at M.U.C. Women's College, Burdwan. His research interests include algorithmics and bioinformatics.

How to cite this paper: Akash Nag,"A Fast Heuristic Algorithm for Solving High-Density Subset-Sum Problems", International Journal of Mathematical Sciences and Computing(IJMSC), Vol.3, No.2, pp. 55-61, 2017.DOI: 10.5815/ijmsc.2017.02.05