Design and Evaluation of Alternate Enumeration Techniques for Subset Sum Problem

Avni Verma and Kamalakar Karlapalem
avni.verma@research.iiit.ac.in, kamal@iiit.ac.in

Data Science and Analytics Center (DSAC)
International Institute of Information Technology, Hyderabad, India

Abstract. The subset sum problem, also referred as SSP, is a NP-Hard computational problem. SSP has its applications in broad domains like cryptography, number theory, operation research and complexity theory. The most famous algorithm for solving SSP is Backtracking Algorithm which has exponential time complexity. Therefore, our goal is to design and develop better alternate enumeration techniques for faster generation of SSP solutions. Given the set of first $n$ natural numbers which is denoted by $X_n$ and a target sum $S$, we propose various alternate enumeration techniques which find all the subsets of $X_n$ that add up to sum $S$.

In this paper, we present the mathematics behind this exponential problem. We analyze the distribution of power set of $X_n$ and present formulas which show definite patterns and relations among these subsets. We introduce three major distributions for power set of $X_n$: Sum Distribution, Length-Sum Distribution and Element Distribution. These distributions are preprocessing procedures for various alternate enumeration techniques for solving SSP. We propose novel algorithms: Subset Generation using Sum Distribution, Subset Generation using Length-Sum Distribution, Basic Bucket Algorithm, Maximum and Minimum Frequency Driven Bucket Algorithms and Local Search using Maximal and Minimal Subsets for enumerating SSP.

We compare the performance of these approaches against the traditional backtracking algorithm. The efficiency and effectiveness of these algorithms are presented with the help of these experimental results. Furthermore, we studied the over solution set of subsets generated by various algorithms to get the complete solution for subset sum problem. Finally, we present a conjecture about upper bound on the number of subsets that has to be enumerated to get all solutions for Subset Sum Problem.

1 Introduction

In SSP, we consider a set of $n$ positive integers stored in set $X$ and a target sum $S$. $X = \{x_1, x_2 \ldots x_n\}$. Traditionally, there are two definitions for SSP which are described below:

1. Version 1: Given a set $X$ containing positive integers and a target sum $S$, is there a subset which sum upto $S$? This is a NP-Complete problem.
   For example, given $X = \{5, 4, 9, 11\}$ and $S = 9$, the solution to this problem is true. There are many ways to solve this problem and it depends on the size and values of $X$ and $S$. The brute force algorithm iterates through all possibilities and takes $\mathcal{O}(2^n \times n)$ time for execution. For smaller size and values of $X$ and $S$, SSP can be solved in polynomial time by using dynamic programming with time complexity $\mathcal{O}(n \times S)$ [19].

2. Version 2: Given a set $X$ containing positive integers and a target sum $S$, find a subset which can sum up to $S$. This is a NP-Hard problem.
   For $X = \{5, 4, 9, 11\}$ and $S = 9$, the solution to above problem is either $\{5, 4\}$ or $\{9\}$. This is an exponential time taking problem which can be solved in $\mathcal{O}(2^n \times n)$ time by using brute force. This method requires $\mathcal{O}(n)$ storage space to store the required result. This version of SSP does not have any known polynomial time algorithm.

In this paper, we extend the traditional SSP (Version 2) and design various alternate enumeration techniques. Instead of finding one subset with target sum, we find all possible solutions of SSP. Therefore, for $X = \{5, 4, 9, 11\}$ and $S = 9$, solutions to our version of SSP are $\{5, 4\}$ and $\{9\}$. We further confine and refine our problem domain by considering first $n$ natural numbers as set $X$. There are many advantages for selecting this problem domain. It simplifies the problem statement, avoids duplication and since sum of
first $n$ natural number is $\frac{n(n+1)}{2}$, by selecting $X = \{1, 2 \ldots n\}$ we restrict target sum between $1$ and $\frac{n(n+1)}{2}$, $S \in [1, \frac{n(n+1)}{2}]$. The efforts to solve Subset Sum Problem are required to get subset queries in relational databases \[18\]. Before describing the formulation of our problem in detail we explore the research work conducted in field of SSP.

2 Related Work

The Subset Sum Problem has been studied very widely. It has a standard $O(nm)$ pseudo-polynomial time dynamic programming algorithm \[17\] which is taught in elementary algorithms class. Additionally, there are a number of other algorithms in the literature, including an FPTAS \[10\], an exact algorithm with space and time trade offs \[11\], a polynomial time algorithm for most low density sums \[14\], and a number of more specialized pseudo-polynomial time algorithms with various properties \[12\] \[5\] \[15\] \[13\].

There is also another variant of Subset Sum Problem which allows the elements in $X$ to be used any number of times in the sums. Overall, dynamic programming is expected to be most efficient for very dense instances, while backtracking is expected to be most efficient for sparse instances of Subset Sum Problem. In different versions of the SSP, the input set may or may not contain duplicate values, and the problem can also be expressed as an optimization problem.

In \[11\], the authors have introduced a new faster pseudo-polynomial time algorithm for the Subset Sum problem to decide if there exists a subset of a given set $S$ whose elements sum to a target number $t$. Their proposed algorithm runs in $O(\sqrt{n}t)$ time, where $n$ is the size of set $S$. Their approach is based on a fast Minkowski sum calculation that exploits the structure of subset sums of small intervals.

Despite the apparent simplicity of the problem statement, to date there has been modest progress on exact algorithms\[4\] for Subset Sum Problem. Indeed, from a worst-case performance perspective the fastest known algorithm runs in $O(2^{\frac{n}{2}})$ time and dates to the 1974 work of Horowitz and Sahni\[9\]. Improving the worst-case running time is a well-established open problem\[20\].

In \[2\], the authors present a randomized algorithm. They consider positive integers and a target sum but instead of finding all subsets of target sum, the solution is bounded by $B$ concentration. The main result of this algorithm is that all instances without strong additive structure (without exponential concentration of sums) can be solved faster than the Horowitz-Sahni time bound $O(2^{\frac{n}{2}})$ \[9\]. They have also shown a quantitative claim to show or prove it. Complexity of this randomized algorithms is $O(2^{0.3399n}B^4)$.

Beier and Vocking \[3\] presented an expected polynomial time algorithm for solving random knapsack instances. Knapsack and subset sum have similarities, but the random instances considered there are quite different from ours, and this leads to the development of quite a different approach. Subset sum problem is also closely related to the classical number theory study of determining partitions. In \[8\] Hardy and Wright provide generating functions but is limited due to lack of computational scheme for generating such partitions. A survey of algorithms for the different variations of the knapsack problem is given in \[8\]. Much of the early work in the knapsack problem was done by Gilmore and Gomory \[6\] \[7\].

However, there is very little work done on enumeration techniques for subset sum problem, which we addressed in this work. We have developed different algorithms for alternate enumerations techniques for subset sum problem and have compared their performance.

3 Formulation for Subset Sum Problem

The following set of information is used for presenting the exponential aspect and solution of alternate enumeration techniques of SSP:

1. A set of first $n$ natural numbers. $X_n = \{1, 2, 3 \ldots n\}$ where $n$ is a positive integer. The set $X_n$ is also known as the Universal set. This is our problem domain. The cardinality of the set $X_n$ is $n$. $|X_n| = n$
2. A set of all subsets of $X_n$ is $P (X_n) = \{\phi, \{1\}, \{2\} \ldots \{1, 2 \ldots n\}\}$. It is also known as power set. The empty set is denoted as $\phi$ or $\{\}$ or the null set. In this paper, we use $\phi$ for the representation. $|P (X_n)| = a = 2^n$
3. $\text{maxSum}(n)$ is the sum of all elements of the universal set $X_n$. This is the maximum possible sum for any element of $P (X_n)$.

$$\text{maxSum}(n) = b = (1 + 2 + 3 \ldots n) = \frac{n(n+1)}{2}$$

$$\text{Sum}(A) \leq \text{maxSum}(n) = \frac{n(n+1)}{2} \forall A \in P (X_n)$$
4. Sum(A) is the sum of all elements of a set A where A belongs to power sets of X_n, A ∈ P(X_n).
   - We assume sum of all elements of φ as 0, Sum(φ) = 0.
   - The range of Sum(A) is [0, n(n+1)/2).
   - The minimum possible sum for A, where A ∈ P(X_n), is denoted as minSum(n).

5. midSum(n) is the mid point of the range of Sum(A) where A ∈ P(X_n). Since, the maximum possible sum for power sets of X_n, P(X_n) is n(n+1)/2 and minimum possible sum is 0, midSum(n) = \[\frac{\text{minSum}(n)+\text{maxSum}(n)}{2}\]
   \[\text{midSum} = d = \frac{(1+2+3...n)}{2} = \frac{n(n+1)}{4}\]
   For simpler calculations, we consider midSum as the largest integer less than or equal to the mid point, floor(midSum(n)) = \[\lfloor \frac{n(n+1)}{4}\rfloor\].

6. maxLen(n) is the count of all elements of the universal set X_n. This is the maximum possible length for any element of P(X_n).
   - Therefore, maxLen(n) is equal to the cardinality of set X_n, defined in point-1.
   - maxLen(n) = |X_n| = |\{1, 2...n\}| = n

7. Len(A) is the count of all elements of a set A where A belongs to power sets of X_n, A ∈ P(X_n).
   - The range of Len(A) is from 1 to n, Len(A) ∈ [1, n].
   - We consider, count of all elements of subset φ as 1, Len(φ) = 1.
   - Therefore, the range of Len(A) is from 1 to n. Len(A) ∈ [1, n].
   - The minimum possible length for A, where A ∈ P(X_n), is denoted as minLen(n).

8. minSum(n, l) is the sum of a subset A where A ∈ P(X_n) with Len(A) = l. A is the subset of length l with minimum possible sum. Subset of length l with minimum possible sum contains first l smallest natural numbers. Therefore, minimum possible subset of length l is A = {1, 2...l}.
   \[\text{minSum}(n, l) = (1+2+...+l) = \frac{l(l+1)}{2}\]

9. maxSum(n, l) is the sum of a subset A where A ∈ P(X_n) and Len(A) = l. A is the subset of length l with maximum possible sum. Subset of length l with maximum possible sum will contain l largest natural numbers decreasing from n.
   - Maximum possible subset of length l is A, A = {n, n-1...n-(l-1)}.
   - maxSum(n, l) = (n + (n-1) + ... + n - l + 1) = n x l - \[\frac{l(l-1)}{2}\]
   - maxSum(n, l) = \[\frac{l(2n-l+1)}{2}\]

4 Distribution Formulae

We have analyzed the distribution of P(X_n) over sum, length and count of individual elements. We present distribution formulas and algorithms, along with example, which show definite patterns and relations among these subsets.

In Table\[\] we briefly present the formula, definition, meaning, values and assumptions of all distributions which are required for design and evaluation of alternate enumeration techniques for SSP. Cardinality of a set is the number of elements of the set. These distributions are prepossessing procedures which are required for presenting our novel alternate enumeration techniques for solving SSP. The formulae are the notation developed in Section\[\]. In Table\[\] b denotes the maximum possible sum for any element of P(X_n), b = \[\frac{n(n+1)}{2}\].

| Distribution | Formula | Meaning | Value/Assumption |
|--------------|---------|---------|-----------------|
| SD Sum-Distribution | n × b, where | SD(n)[S] represents the count of all the subsets belonging to P(X_n) with sum S. Every row, SD[n], is the sum distribution for all subsets of X_n where sum is S. | In this thesis, the empty set φ is counted once while calculating the sum distribution, SD[n][0] = 1. |
4.1 Sum Distribution

In sum distribution, also referred as $SD$, we find the number of subsets which sum up to a certain integer $S$, where $X_n = \{1, 2, 3 \ldots n\}$ and $S \in [0, \frac{n(n+1)}{2}]$. It is represented as $SD[n][S]$. Equation (1) establishes the formula for the sum distribution. Before counting the subsets of a particular sum, we initialize the count as zero, $\forall n, S SD[n][S] = 0$. Following are the base cases for sum distribution ($SD[n][S]$):

1. For $n = 0$ and $S = 0$, the corresponding subset is $\phi$. Since, zero-sum ($Sum = 0$) can be achieved only with subset $\phi$ and $Sum(\phi)$ is assumed to be 0, as defined in Section 4.3, the count of occurrence of $\phi$-subset in $P(X_0)$ is taken as 1. Therefore, $SD[0][0] = 1$.

2. $\forall i \in [1, n]$ and $S = 0$, $SD[i][0] = 1$. Since, zero-sum ($Sum = 0$) can be achieved only with subset $\phi$, the count of occurrence of $\phi$-subset in $P(X_n)$ is taken as 1. Therefore, $SD[n][0] = 1$.

3. $SD[i][j] = 0$, if $i < 0$ or $j < 0$.

$$SD[n][S] = \begin{cases} 
1 & (S = 0) \text{ or } (n = 1) \\
0 & (n = 0) \\
SD[n-1][S] & 0 < S < n \\
SD[n-1][S] + SD[n-1][S-n] & n \leq S \leq \left\lfloor \frac{n(n+1)}{4}\right\rfloor \\
SD[n][maxSum(n) - S] & \left\lfloor \frac{n(n+1)}{4}\right\rfloor < S \leq maxSum(n) = \frac{n(n+1)}{2} \\
0 & \text{otherwise}
\end{cases} \quad (1)$$

Similar to Element Distribution (Section 4.3), we can give uniqueness and correctness proof of Sum Distribution.
4.2 Length-Sum Distribution

In length-sum distribution, we find the number of subsets of \( X_n \) of length \( l \) which sum up to \( S \) where \( S \in [0, \text{maxSum}(n)] \), \( \text{maxSum}(n) = \frac{n(n+1)}{2} \) and \( l \in [0, n] \). Table 2 presents the bases cases for Length-Sum Distribution.

\[
LD[n][S][l] = \begin{cases} 
1 & l = 0 \text{ and } S = 0 \\
LD[n-1][S][l] & 1 \leq l \leq \left\lfloor \frac{n}{2} \right\rfloor \text{ and } 0 \leq S < n \\
LD[n-1][S-l][n-1] + LD[n-1][S-S-n][l-1] & 1 \leq l \leq \left\lfloor \frac{n}{2} \right\rfloor \text{ and } n \leq S \leq \frac{n(n+1)}{2} \\
LD[n][\text{maxSum}(n) - S][n-l] & \left\lfloor \frac{n}{2} \right\rfloor < l \leq n \\
0 & \text{otherwise}
\end{cases}
\]

(2)

Values of \( l \) for \( n = 0 \):

| Values of \( l \) for \( n = 0 \) | Subset | Sum of the Subset | No. of Subsets / Length Distribution |
|-------------------------------|------|-----------------|-------------------------------------|
| \( l=0 \) | \{\phi\} | 0 | 1 |

Values of \( l \) for \( n = 1 \):

| Values of \( l \) for \( n = 1 \) | Subset | Sum of the Subset | No. of Subsets / Length Distribution |
|-------------------------------|------|-----------------|-------------------------------------|
| \( l=0 \) | \{\phi\} | 0 | 1 |
| \( l=1 \) | \{1\} | 1 | 1 |

Values of \( l \) for \( n = 2 \):

| Values of \( l \) for \( n = 2 \) | Subset | Sum of the Subset | No. of Subsets / Length Distribution |
|-------------------------------|------|-----------------|-------------------------------------|
| \( l=0 \) | \{\phi\} | 0 | 1 |
| \( l=1 \) | \{1\} | 1 | 1 |
| \( l=2 \) | \{1, 2\} | 3 | 1 |

Table 2: Length-Sum Distribution for base cases: \( X_0, X_1 \) and \( X_2 \). First column presents the possible length values, second and third column presents the corresponding subsets and their sum respectively and the fourth column presents the Length-Sum distribution, \( LD[n][S][l] \).

4.3 Element Distribution

In Section 4.1, we have explained and explored the concept of Sum Distribution, where we count the number of subsets out of all power set \( P(X_n) \), of \( X_n \) which add up to a certain number \( S \). Let us assume, \( M \) represents such sets. We study the occurrence of each element from set \( X_n \) in set \( M \). \( e \) denotes each element \( \forall e \in [1, n], \forall S \in [0, \frac{n(n+1)}{2}] \), element distribution function, \( ED[n][S][e] \), is defined as follows:

\[
ED[n][S][e] = \begin{cases} 
0 & (n = 0) \text{ or } (S = 0) \text{ or } (e = 0) \\
ED[n-1][S][e] & 0 \leq S < n \text{ and } 1 \leq e < n \\
ED[n-1][S][e] + ED[n-1][S-n][e] & n \leq S \leq \frac{n(n-1)}{2} \text{ and } 1 \leq e < n \text{ and } n > 2 \\
SD[n-1][S-n] & n \leq S \leq \frac{n(n+1)}{2} \text{ and } e = = n \\
SD[n][S] - ED[n][\text{maxSum} - S][e] & n \leq S \leq \frac{n(n+1)}{2} \text{ and } 1 \leq e < n \\
0 & \text{otherwise}
\end{cases}
\]

(3)

Element distribution is another prepossessing procedure required for presenting various alternate enumeration techniques especially bucket algorithms introduced in Section 5.4.
Table 3: Length-Sum Distribution for $P(X_3)$

Table 4: Distribution of elements $[1,2]$ in $P(X_2)$ and elements $[1,2,3]$ in $P(X_3)$.

Correctness of the Element Distribution Formula

We present the theorems and lemma which prove the correctness of Element distribution formula, $ED[n][S][e]$ presented in Equation 3. $ED[n][S][e]$ represents the count of element $e$ in those subsets of $X_n$ which has sum $S$ where $e \in [1,n]$. $S \in [0,maxSum]$ and $maxSum = \frac{n(n+1)}{2}$.

**Theorem 1** $ED[n][S][n] = 0$ if $0 < S < n$.

**Proof.** Let us assume $ED[n][S][e] \neq 0$ and $ED[n][S][e] = c$, where $c$ is a positive integer. $c$ is the count of number of times an element $e$ occur in a class of subsets $element(n,S,e)$ where $element(n,S,e)$ consist of all
the subsets of $\mathcal{P}(X_n)$ which add up to a sum of $S$. Since, $c$ represents a count, it cannot be negative. By definition $c$, $ED[n][S][e]$ and $element_{(n,S,e)}$ follow these equations:

$$c = |element_{(n,S,e)}|$$

(4)

$$ED[n][S][e] = |element_{(n,S,e)}|$$

(5)

$$c = ED[n][S][e]$$

(6)

Let $A$ be a subset of $element_{(n,S,e)}$. Then, $e$ will belong to $A$ and sum of all elements of $A$ will be greater than or equal to $e$.

$$e \in A$$

(7)

$$Sum(A) \geq e$$

(8)

$$S \geq e$$

(9)

Since ($e == n$) as per the initial conditions, Equation 9 will become,

$$S \geq n$$

(10)

Since $0 \leq S < n$ it results into a contradiction. Our assumption is false. There are no subsets which contain $e$ and have sum less than $e$. Therefore, from the condition $c = 0$ and from Equation 9

$$ED[n][S][e] = 0$$

(11)

Hence, we have proved the first part of Equation 3.

**Theorem 2** $ED[n][S][e] = ED[n-1][S][e] + ED[n-1][S-n][e]$ if $n \leq S \leq \frac{n(n-1)}{2}$, $1 \leq e < n$ and $n > 2$.

**Proof.** Let $element_{(n,S,e)}$ be a class of subsets which consists of all the subsets of $P(X_n)$ which sum up to $S$ and contain an element $e$, $1 \leq e < n$. Let us assume, a set $A \in element_{(n,S,e)}$. Since ($S \geq n$), then $A$ may or may not contain element $n$. If $n \in A$ then $A - n$ belongs to the class of subsets of $P(X_{n-1})$ which sum up to $(S - n)$ and contain an element $e$ (as presented in Equation 13). If $n \notin A$ then, $A$ belongs to the class of subsets of $P(X_{n-1})$ which sum up to $S$ and contain an element $e$ (as presented in Equation 14).

$$A - n \in element_{(n-1,S-n,e)}$$

(13)

$$A \in element_{(n-1,S,e)}$$

(14)
From Equation 13 and Equation 14 we form the set of all subsets which sum up to $S$ and contain element $e$.

\[ \text{element}_{(n,S,e)} = \text{element}_{(n-1,S,e)} \cup \text{element}_{(n-1,S-n,e)} \quad n \leq S \leq \frac{n(n-1)}{2} \quad \text{and} \quad 1 \leq e < n \]  
(15)

Taking cardinality on both sides of Equation 15

\[ |\text{element}_{(n,S,e)}| = |\text{element}_{(n-1,S,e)}| + |\text{element}_{(n-1,S-n,e)}| \quad n \leq S \leq \frac{n(n-1)}{2} \quad \text{and} \quad 1 \leq e < n \]  
(16)

Thus, $ED[n][S][e] = ED[n-1][S][e] + ED[n-1][S-n][e]$ $n \leq S \leq \frac{n(n-1)}{2} \quad \text{and} \quad 1 \leq e < n$  
(17)

In order to complete this proof following properties of $\text{element}_{(n,S,e)}$ should be proved.

1. **Uniqueness:** There should be no duplicate subsets in $\text{element}_{(n,S,e)}$.

   **Proof:** $\text{element}_{(n-1,S,e)}$ is the set of all the subsets of $\mathcal{P}(X_{n-1})$ containing element $e$ with sum $S$ and $\text{element}_{(n-1,S-n,e)}$ is the set of all the subsets of $\mathcal{P}(X_{n-1})$ containing element $e$ with sum $(S-n)$. We use the method of contradiction to prove set of subsets in $\text{element}_{(n-1,S,e)}$ and $\text{element}_{(n-1,S-n,e)}$ are independent. Let us assume, subset $p$ belongs to both $\text{element}_{(n-1,S,e)}$ and $\text{element}_{(n-1,S-n,e)}$. Since, $p \in \text{element}_{(n-1,S,e)}$, therefore by definition, the subset $p$ contains element $e$, has elements ranging from 1 to $(n-1)$ and these elements sum upto $S$.

   \[ S = \sum_{i=1}^{\text{len}(p)} p_i \]  
(18)

Similarly, as per assumption, $p \in \text{element}_{(n-1,S-n,e)}$. Therefore by definition, the subset $p$ contains element $e$, has elements ranging from 1 to $(n-1)$ and these elements sum upto $(S-n)$.

\[ (S-n) = \sum_{i=1}^{\text{len}(p)} p_i \]  
(19)

From Equation 18 and Equation 19 there is a contradiction as $\sum_{i=1}^{\text{len}(p)} p_i$ is both $S$ and $(S-n)$. Since, $n$ is a natural number, the above equations contradict our assumption that a subset $p$ can belong to both sets $\text{element}_{(n-1,S,e)}$ and $\text{element}_{(n-1,S-n,e)}$. Therefore, by contradiction, there is no subsets $p$ which belongs to both sets. Hence, $\text{element}_{(n-1,S,e)}$ and $\text{element}_{(n-1,S-n,e)}$ are independent.

2. **Completeness:** $\text{element}_{(n,S,e)}$ should contain all the subsets of $\mathcal{P}(X_n)$ which contain element $e$ and sum upto $S$.

   **Proof:** The power set of $X_n$, $\mathcal{P}(X_n)$ which contain element $e$ and sum upto $S$ can be divided into two parts: subsets with sum $S$ which contain element $n$ and subsets with sum $S$ which do not contain element $n$. By definition, $\text{element}_{(n-1,S,e)}$ is the set of all the subsets of $\mathcal{P}(X_{n-1})$ with sum $S$ containing element $e$ and $\text{element}_{(n-1,S-n,l-1)}$ is the set of all the subsets of $\mathcal{P}(X_{n-1})$ with sum $(S-n)$ containing element $e$.

   In Equation 15 the union of sets $\text{element}_{(n-1,S,e)}$ and $\text{element}_{(n-1,S-n,e)}$ generates all subsets of $\mathcal{P}(X_n)$ with sum $S$ containing element $e$. Therefore, $\text{element}_{(n,S,e)}$ should consists of subsets of $\mathcal{P}(X_n)$ with sum $S$ containing element $e$.

   The above two proofs are required to complete the statement: $ED[n][S][e] = ED[n-1][S][e] + ED[n-1][S-n][e]$ if $n \leq S \leq \frac{n(n-1)}{2}$ and $1 \leq e < n$. This theorem will only be true, if sum is positive i.e. $S \geq 0$

\[ S \geq 0 \]  
(20)

\[ \frac{n(n-1)}{2} - n \geq 0 \]  
(21)
\[
\frac{n^2 - n - 2n}{2} \geq 0 \quad (22)
\]
\[
\frac{n^2 - 3n}{2} \geq 0 \quad (23)
\]
\[
\frac{n(n - 3)}{2} \geq 0 \quad (24)
\]
\[
n(n - 3) \geq 0 \quad (25)
\]

Therefore, either both \(n\) and \(n - 3\) should be greater than 0 or both should be less than 0. Since, \(n\) cannot be negative,
\[
n \geq 0 \quad \text{and} \quad n \geq 3 \quad (26)
\]
Therefore,
\[
n \geq 3 \quad (27)
\]

Hence, from Equation (17) and Equation (27) we have proved the third part of Equation (3).

**Lemma 3** \(ED[n][S][e] = ED[n - 1][S][e]\) if \(0 \leq S < n\) and \(1 \leq e < n\).

**Proof.** According to Theorem 2,
\[
ED[n][S][e] = ED[n - 1][S][e] + ED[n - 1][S - n][e] \quad n \leq S \leq \frac{n(n - 1)}{2} \quad \text{and} \quad 1 \leq e < n \quad (28)
\]
Since,
\[
0 \leq S < n \quad (29)
\]
\[
(-n) \leq S - n < 0 \quad (30)
\]
But a sum cannot be negative. Therefore, count of element \(e\) in subsets of \(\mathcal{P}(X_{n-1})\) which sum up to \(S\) is zero, \(ED[n - 1][S - n][e] = 0\).
\[
ED[n][S][e] = ED[n - 1][S][e] + 0 \quad (31)
\]
\[
ED[n][S][e] = ED[n - 1][S][e] \quad 0 \leq S < n \quad \text{and} \quad 1 \leq e < n \quad (32)
\]
Equation (32) proves the second part of Equation (3).

**Theorem 4** \(ED[n][S][e] = SD[n - 1][S - n]\) if \(n \leq S \leq \frac{n(n + 1)}{2}\) and \(e = n\).

**Proof.** Let \(\text{element}_{(n,S,e)}\) be a class of subsets where it consist of all the subsets of \(\mathcal{P}(X_{n})\) which sum up to \(S\) and contain an element \(e, e = n\). Let us assume \(A \in \text{element}_{(n,S,e)}\) and \(|\text{element}_{(n,S,e)}| = ED[n][S][e] = c\) where \(c \geq 0\). Since, element \(e\) belongs to set \(A, e \in A,\)
\[
A - e \equiv A - n \in \text{element}_{(n-1,S-e,0)} \quad (33)
\]
Sum \(S\) will result in following condition,
\[
n \leq S \leq \frac{n(n + 1)}{2} \quad (34)
\]
\[
0 \leq S - n \leq \frac{n(n + 1)}{2} - n \quad (35)
\]
Let us assume \(S - n\) as \(S'\),
\[
0 \leq S' \leq \frac{n^2 - n}{2} \quad (36)
\]
\[ 0 \leq S' \leq \frac{n(n-1)}{2} \] (37)

\[ \maxSum(n-1) = \frac{n(n-1)}{2} \] (38)

From Equation 33 and Equation 38

\[ \forall A - n \in \text{element}_{(n-1,S-n,0)} = \text{element}_{(n-1,S-n,e')} \quad \text{where} \quad 1 \leq e' < n \] (39)

\[ \forall A \in \text{element}_{(n,S-n+n,n)} \equiv \text{element}_{(n-1,S-n,e')} \quad \text{where} \quad 1 \leq e' < n \] (40)

\[ \forall A \in \text{element}_{(n,S,n)} \equiv \text{element}_{(n-1,S-n,e')} \quad \text{where} \quad 1 \leq e' < n \] (41)

Taking cardinality on both sides,

\[ |\text{element}_{(n,S,n)}| = |\text{element}_{(n,S,r)}| = |\text{element}_{(n-1,S-n,e')}| \] (42)

\[ ED[n][S][e == n] = ED[n-1][S-n][e'] \] (43)

\[ |\text{element}_{(n-1,S-n,e')}| \] is the number of subsets \(X_{n-1}\) which sum up to \((S - n) = (S - e) = (S - n)\). By using the concept of sum distribution defined in Section 4.1 and Equation 43

\[ ED[n-1][S-n] \] (44)

\[ ED[n-1][S-e'] = ED[n][S][e == n] = SD[n-1][S-n] \] (45)

\[ ED[n][S][e] = SD[n-1][S-n] \quad \text{where} \quad n \leq S \leq \frac{n(n+1)}{2} \quad \text{and} \quad e == n \] (46)

Equation 46 proves the fourth part of Equation 3.

**Theorem 5** \( ED[n][S][e] = SD[n][S] - ED[n][\maxSum(n) - S][e] \) if \((\frac{n(n-1)}{2} + 1) \leq S \leq \frac{n(n+1)}{2} \) and \(1 \leq e < n\)

*Proof.* \( \maxSum(n) \) is the sum of all elements of \(X_n = 1 + 2 + \ldots + n = \frac{n(n+1)}{2} \), as defined in Section 3. Let us assume \( S' = \maxSum(n) - S \). Since, the plot between number of subsets and sum follow a Gaussian symmetric distribution, \( SD[n][S] \) will be equal to \( SD[n][\maxSum(n) - S] \).

\[ SD[n][S] = SD[n][S'] = c \] (47)

There are \( c \) number of subsets which sum up to \( S \) and \( S' \). In this case, sum \( S \) is greater than the \( \maxSum \) (the mid point) and by using the reflection/symmetric property of the curve we can find all the values of \( ED[n][S][e] \).

\[ S'' = \frac{\maxSum}{2} = \frac{n(n+1)}{4} \] (48)

\[ S_{low} = \frac{n(n-1)}{2} + 1 \] (49)

\[ S_{low} - S'' = \frac{n(n-1)}{2} + 1 - \frac{n(n+1)}{4} \] (50)

\[ S_{low} - S'' = \frac{n(2n-2-n+1)}{2} + 1 \] (51)

\[ f(n) = S_{low} - S'' = \frac{n^2 - 3n + 2}{2} \] (52)
By using the property of second derivative test we show that \( f'(n) \) is greater than 0 when \( S > \frac{\text{maxSum}}{2} \).

\[
f'(n) = \frac{d(f(n))}{dn} > 0 \tag{53}
\]

\[
f'(n) = \frac{d(n^2 - 3n + 2/2)}{dn} > 0 \tag{54}
\]

\[
f'(n) = n - \frac{3}{2} > 0 \tag{55}
\]

\[
f'(n) = n > \frac{3}{2} \tag{56}
\]

Therefore, \( \forall n \geq 2 \) we can use the symmetric property and calculate half of the values by using the previously calculated values. For \( n = 1 \) values of element distribution will be covered as the part of base cases.

Let \( \text{sum}_{(n,S)} \) be a set of all the subsets of \( \mathcal{P}(X_n) \) which sum up to \( S \) and \( \text{sum}_{(n,S')} \) consist of all subsets of \( \mathcal{P}(X_n) \) which sum to \( S' \), where \( S' = (\text{maxSum} - S) \), \( \forall A \in \text{sum}_{(n,S)} \) and \( A^c \in \text{sum}_{(n,S')} \) where \( A^c \) is the complement set of \( A \).

\[
A \cup A^c = U \tag{57}
\]

Since, \( U \) is the universal set, \( U = \{1, 2 \ldots n\} \) and contain a single occurrence of each element \( e \in [1, n] \), therefore, \( A \cup A^c \) also contains a single occurrence of each element \( e \). From Equation (57) there are \( c \) subsets in \( A \) and \( A^c \). \( \forall k \in [1, n] \) count of \( e \) in \( A \) and \( A^c \) is 1. Let us define \( \text{Count}(x, y) \) as the count of element \( x \) in any subset or class of subsets \( y \).

\[
\text{Count}(e, A) + \text{Count}(e, A^c) = 1 \tag{58}
\]

\[
\forall e \in [1, n], \forall A \in \text{sum}_{(n,S)} \text{ and } \forall A^c \in \text{sum}_{(n,S')} \text{ }
\]

\[
\text{Count}(e, \text{sum}_{(n,S)}) + \text{Count}(e, \text{sum}_{(n,S')}) = |\text{sum}_{(n,S)}| \times 1 = |\text{sum}_{(n,S')}| \times 1 \tag{59}
\]

By using the definition of element distribution and Equation (47)

\[
\text{ED}[n][S][e] + \text{ED}[n][S'][e] = 1 \tag{60}
\]

\[
\text{ED}[n][S][e] + \text{ED}[n][S'][e] = \text{SD}[n][S] \tag{61}
\]

\[
\text{ED}[n][S][e] = \text{SD}[n][S] - \text{ED}[n][S'][e] \tag{62}
\]

Therefore, by putting the value of \( S' = (\text{maxSum}(n) - S) \)

\[
\text{ED}[n][S][e] = \text{SD}[n][S] - \text{ED}[n][\text{maxSum}(n) - S][e] \tag{63}
\]

Equation (63) proves the last part of Equation (3).

Sum Distribution, Length-Sum Distribution and Element Distribution are used in developing alternate enumeration techniques for solving SSP. These techniques are presented in the next section.

5 Alternate Enumeration Techniques for Subset Sum Problem

In this paper, we propose seven approaches to find the solution for enumerating all the \( (2^n - 1) \) subsets of \( X_n \). In each approach, we choose different method for addressing the enumeration of SSP. We propose novel algorithms: Subset Generation using Sum Distribution (SDG), Subset Generation using Length-Sum Distribution (LDG), Basic Bucket Algorithm (Basic BA), Maximum and Minimum Frequency Driven Bucket Algorithms (Max FD and Min FD) and Local Search using Maximal and Minimal Subsets (LS MaxS and LS MinS) for enumerating SSP. The first approach is the backtracking algorithm. It is the naive method for solving SSP. This algorithm is used to benchmark the new proposed algorithms.
5.1 Subset Generation using Backtracking

Our aim is to find all the subsets of set $X_n$ with $\text{Sum} = S$. According to the exhaustive search algorithm for SSP [19], we try to find the resulting subset by iterating through all possible $2^n$ solutions. But in this algorithm, we arrange the elements in an orderly fashion. The worst case time complexity for this algorithm is exponential. It is $O(n \times 2^n)$. The space complexity for this algorithm is the size of the input, $O(n)$. Even though backtracking is a clean and crisp algorithm for SSP, this algorithm has many drawbacks. It tries to generate all the desired subsets by checking every branch and subset. Since there can be a lot of high branches at every state of the back tracking algorithm, this leads to inefficient, multiple recursive calls and reversion to old states. It requires a large amount of time and space to reflect the changes in the system stack.

5.2 Subset Generation using Sum Distribution

We design a generator using Sum Distribution. Algorithm 1 is the pseudo-code for generating all the subsets of set $X_n$ with sum $S$. As we know, sum distribution is recursive and uses subsets of $X_{(n-1)}$ to produce results for $X_n$. We store these previous values with the help of $SDG$ (initialized at Line 1). Extra values of $SDG$ ($SDG[i-1]$) are freed in Line 20 to minimize the space consumption. In Line 2, we iterate through smaller universal numbers. Line 3 to Line 6 define $start\_sum$, $mid\_sum$, $end\_sum$ and $universal\_set$. Line 7 to Line 19 iterate through values of sum between $start\_sum$ and $mid\_sum$. The desired set of subsets, $SDG[i][j]$ (subsets of $X_i$ with sum $j$), consists of all subsets of $SDG[i-1][j]$ and $SDG[i-1][j-i]$. Next, we include $i^{th}$ element in every subset of $SDG[i-1][j-i]$. For each of these resulting subsets, a symmetric subset of sum $(end\_sum − j)$ is calculated by subtracting the subset from $universal\_set$. Line 11 to Line 18 essentially execute these steps and returns the final result at Line 22.

The value of maximum number of subsets has exponential bound, $O(2^n \times n^{\frac{n-1}{2}})$, as described in Appendix 1. Therefore, the time complexity for (loop1) at Line 14 is $O(2^n \times n^{\frac{n-1}{2}})$. Since, $n \in [1, n]$ and $S \in [0, \frac{n(n+1)}{2}]$, time complexity of the above algorithm results to $O(loop1) \times O(loop2) \times O(loop3) = O(n) \times O(n^2) \times O(2^n \times n^{\frac{n-1}{2}}) = O(2^n \times n^{\frac{n}{2}})$. Space complexity for the above algorithm is the size of array storing smaller subsets, $SDG[n-1][S]$. This complexity is also exponential $n \times S \times \text{No. of Subsets}$. Since $S \in [0, \frac{n(n+1)}{2}]$, the space complexity results to $O(n) \times O(n^2) \times O(2^n \times n^{\frac{n}{2}})$ i.e. $O(2^n \times n^{\frac{n}{2}})$.

Algorithm 1 SDG: GeneratorUsingSumDistribution($n$)

1: $SDG = \{\}$ \hspace{1cm} \Comment{Data structure to store the generated Subsets}
2: for $i \in \{1, \ldots, n\}$ do
3:  \hspace{1cm} $start\_sum = 0$
4:  \hspace{1cm} $mid\_sum = \left\lfloor \frac{1 + \binom{i}{1}}{2} \right\rfloor$
5:  \hspace{1cm} $end\_sum = \left\lceil \frac{i(i + 1)}{2} \right\rceil$ \hspace{1cm} \Comment{$end\_sum$ is equal to $maxSum(i)$}
6:  \hspace{1cm} $universal\_set = \{1, 2 \ldots n\}$ \hspace{1cm} \Comment{$universal\_set$ is used to calculate the symmetric subsets}
7: for $j \in \{start\_sum, \ldots, mid\_sum\}$ do
8:  \hspace{1cm} if $(j == 0)$ then
9: \hspace{2cm} $SDG[i] = \{\phi\}$
10:  \hspace{1cm} end if
11: \hspace{1cm} $SDG[i][j] = SDG[i-1][j]$ \hspace{1cm} \Comment{Adding $i^{th}$ element in every subset of $SDG[i-1][j-i]$}
12: \hspace{1cm} for subset $\in SDG[i-1][j-i]$ do
13: \hspace{2cm} subset.append(i)
14: \hspace{1cm} $SDG[i][j].append(subset)$
15: \hspace{1cm} end for
16: \hspace{1cm} if $j \neq (i - j)$ then
17: \hspace{2cm} $SDG[i][end\_sum - j] = universal\_set - SDG[i][j]$ \hspace{1cm} \Comment{Symmetric subsets.}
18: \hspace{1cm} end if
19: \hspace{1cm} end for
20: \hspace{1cm} $Free(SDG[i-1])$
21: end for
22: return $SDG[n]$
5.3 Subset Generation using Length-Sum Distribution

Along with Sum Distribution, we have established several concepts, theories and formulas for Length-Sum Distribution as well. It counts the number of subsets of \( X_n \) of length \( l \) and sum \( S \) where \( X_n = \{1, 2, 3 \ldots n\} \), \( l \in [0, n] \) and \( S \in [0, \frac{n(n+1)}{2}] \), represented by \( LD[n][S][l] \). The recursive equation (Equation 2) establishes the theory for the Length-Sum distribution.

In this section, we present the designed generator. Algorithm 2 is the pseudo-code for generating all the subsets of \( X_n \) of length \( l \) and sum \( S \). This distribution is recursive and uses \( LDG \) to store the previous output which is initialized at Line 1 and Line 10. The notation for \( LDG \) is different than notation of \( LD \). We denote the count the number of subsets of \( X_n \) calculated so far, to construct all the subsets of \( X_n \) with length \( j \) and \( \text{Sum} = k \). In \( LDG \) notation for length and sum are reversed for easier calculations.

In Algorithm 2 extra values of \( LDG \) (\( LDG[i-1] \)) are freed in Line 26 to minimize the space consumption. In Line 5, Line 9 and Line 13, we iterate through smaller natural numbers, length range and possible values of sum respectively. Line 6 to Line 12 we define \( \text{max\_sum} \) for \( X_i \), bases cases of \( LDG[i][j] \), \( \text{start\_sum} \) and \( \text{end\_sum} \). Line 13 to Line 24 iterates through feasible values of sum between \( \text{start\_sum} \) and \( \text{end\_sum} \). The desired set of subsets, \( LDG[i][j][k] \) consists of all subsets of \( LDG[i-1][j][k] \) and \( LDG[i-1][j-1][k-i] \). We include \( \text{ith} \) element in every subset of \( LDG[i-1][j-1][k-i] \). For each of these resulting subsets, a symmetric subset of length \( (i-j) \) and sum \( \text{end\_sum} - k \) is calculated by subtracting the subset from \( \text{universal\_set} \).

Line 15 to Line 23 essentially execute these steps and returns the final result at Line 28.

The value of maximum number of subsets has exponential bound, \( O(2^n \times n^{\frac{3}{2}}) \), as described in Appendix 3. Therefore, the time complexity for (\( \text{loop}_1 \)) in Line 16 is \( O(2^n \times n^{\frac{3}{2}}) \). Since, \( l \in [1, n] \) and \( S \in [0, \frac{n(n+1)}{2}] \) time complexity of the above algorithm results to \( O(\text{loop}_1) \times O(\text{loop}_2) \times O(\text{loop}_3) \times O(\text{loop}_4) \) i.e. \( O(n) \times O(\text{sum}) \times O(2^n \times n^{\frac{3}{2}}) = O(n^2 \times 2^n \times n^{\frac{3}{2}}) = O(2^n \times n^{\frac{3}{2}}) \). Space complexity for the above algorithm is the size of array storing smaller subsets, \( LDG[n-1] \). This complexity is also exponential \( n \times (\text{No. of Subsets}) \).

Since, \( l \in [1, n] \) and \( S \in [0, \frac{n(n+1)}{2}] \) the space complexity results to \( O(n) \times O(n) \times O(n^2) \times O(2^n \times n^{\frac{3}{2}}) = O(n^4) \times O(2^n \times n^{\frac{3}{2}}) \) i.e. \( O(2^n \times n^{\frac{3}{2}}) \).

5.4 Subset Generation using Basic Bucket Algorithm

In this section, we present a new method which generate all the subsets of \( X_n \) with a particular sum. This is a greedy algorithm. The look-up table that has been used, has been explained in Section 4. It has been extensively used with this algorithm.

The core idea behind this enumeration technique is to use the various distribution values that we have calculated so far, to construct all the subsets of \( X_n \) which sum up to \( S \).

**Given:** The first concept used for Basic Bucket Algorithm is Element Distribution. We start with the exact occurrence of each element of \( X_n \) in subsets of precise sum, \( S \). This information is denoted by \( ED[n][S][e] \).

The next concept used is the number of subsets, among power set of \( X_n \), where summation of all elements is \( S \). \( SD[n][S] \) denotes such count. For this algorithm, we consider \( SD[n][S] \) as number of empty buckets. Buckets are storage data structures which are used to stack all the appropriate elements that compute the total sum \( S \). We iterate through all elements in descending order. During each iteration, an element is assigned to one of the buckets. This method is about adding the correct element to the corresponding subset.

**Properties:** Element distribution and below properties help us ensure the correct placement for every element.

1. An element \( e \) is added to a bucket \( b \) only if the addition results to the uniqueness among all existing elements of the bucket \( b \). This property is followed to guarantee that the generated result is a subset and it is not a bag. A subset belongs to power sets of \( X_n \) \( P(X_n) \).
2. An element \( e \) is added to a bucket \( b \) only if the addition of the element results to uniqueness amongst all the buckets. We follow this property to ensure the generation of correct number of subsets of sum \( S \).
3. An element \( e \) is added to a bucket \( b \) only if on adding the new element, the sum of the bucket does not exceed the desired sum \( S \). This property allows us to create subsets of sum \( S \).

Unfortunately, we have no rule which forces only the generation of subsets with sum \( S \). Many subsets with sum less than \( S \) are generated during the first iteration of this technique. These subsets are called the
Algorithm 2: LDG: GeneratorUsingLengthSumDistribution(n)

1: $LDG = \{\}$
2: $LDG[0][0] = LD[1][0] = \{\}$ \Comment{Base Cases}
3: $LD[1][1] = \{[1]\}$ \Comment{Base Cases}
4: $universal\_set = [1, 2 \ldots n]$ \Comment{$universal\_set$ is used to calculate the symmetric subsets}
5: for $i \in \{2, \ldots, n\}$ do
6: \quad $max\_sum = \frac{(i+1)}{2}$
7: \quad $LDG[i][0] = \{[1]\}$
8: \quad $LDG[i][max\_sum] = \{universal\_set\}$
9: \quad for $j \in \{1, \ldots, \frac{i}{2}\}$ do \Comment{Iterating till mid point}
10: \quad \quad $LDG[i][j] = \{\}$
11: \quad \quad $start\_sum = \frac{(i+1)}{2}$
12: \quad \quad $end\_sum = i \ast j - \frac{(i-1)}{2}$
13: \quad \quad for $k \in \{start\_sum, \ldots, end\_sum\}$ do
14: \quad \quad \quad $LDG[i][j][k] = LDG[i - 1][j][k]$ \Comment{Adding $i^{th}$ element in every subset of $LDG[i - 1][j - 1][k - i]$}
15: \quad \quad \quad if $j \geq 1$ and $k \geq i$ and $i \leq k < \frac{(i+1)}{2}$ then
16: \quad \quad \quad \quad for subset $\in LDG[i][j - 1][k - i]$ do
17: \quad \quad \quad \quad \quad subset.append(i)
18: \quad \quad \quad \quad end for
19: \quad \quad \quad end if
20: \quad \quad end for
21: \quad \quad if $j \neq (i - j)$ then
22: \quad \quad \quad $LDG[i][i - j][end\_sum - k] = universal\_set - LDG[i][j][k]$ \Comment{Symmetric subsets}
23: \quad \quad end if
24: \quad end for
25: \quad $Free(LDG[i - 1])$
27: end for
28: Return $LD[n]$
undesired set. For every subset of the undesired set, \( A \sum(A) \) is less than \( S \) i.e. \( \sum(A) < S \). Therefore, we have converted this technique to a greedy algorithm. Instead of using this as a one time procedure, we reapply it with modified values of element distribution, \( ED[n][S][e] \) and sum distribution, \( SD[n][S] \). All subsets are generated by applying the same technique on modified input in a greedy manner.

**Uniqueness:** The key step in successfully generating the full desired results is to maintain an efficient and complete lookup table as described in Section 7. This lookup table which is maintained with the help of a hash function and bit vectors, not only ensures uniqueness among and within the buckets but also makes sure that all the undesired subsets of previous iterations are properly hashed. So, we do not re-generate the same undesired set in the next iteration. We need to put extra effort to preserve the state of all undesired sets from every iteration. The whole lookup table is no bigger than \( 2^n \) and every subset: desired or undesired, is stored in the form of one integer \( \text{num} \), where \( \text{num} \in [0, 2^n] \). With the aim of preserving the count of every element from the set \( X_n \), we maintain a log table for each round of iterations. The value of log table for each element, at the start of every round is the summation of value of element distribution at the end of last iteration of previous round and the count of all these elements from buckets which do not provide a subset of desired sum.

Algorithm 3 calculates the element distribution before start of each round of Basic Bucket Algorithm. Algorithm 4 initializes the buckets at the start of the algorithm. It finds the value of \( x \) and accordingly fills the buckes with the starting elements. This method is called from Line 3 of the function \textit{GENERATING-SUBSETS}(n, S, SD[n][S], ED[n][S], prevWrongSubsets) from the main Algorithm 6. We find an appropriate bucket for every element based on the properties of the Basic Bucket Algorithm. Functionality is defined in Algorithm 5. While Algorithm 7 iterates though all the rounds of the bucket algorithm. All iterations of every round is implemented by the Algorithm 6.

---

**Algorithm 3** Basic BA: GetED(n, S, Table, wrongSubsets)

1: function \text{GetED}(n, S, Table, wrongSubsets)
2: \hspace{1em} newTable = Table
3: \hspace{1em} for subset \in wrongSubsets do
4: \hspace{2em} for ele \in subset do
5: \hspace{3em} newTable[ele]++ = 1 \footnote{Restoring all the elements of wrongSubsets to the element distribution}
6: \hspace{2em} end for
7: \hspace{1em} end for
8: Return newTable
9: end function

**Algorithm 4** Basic BA: InitializeBuckets(all_buckets, Table, n, S, p)

1: function \text{InitializeBuckets}(all_buckets, Table, n, S, p)
2: \hspace{1em} q = count of non-zero entries of Table
3: \hspace{1em} x = \min(p, q)
4: \hspace{1em} elements = x largest integers of \( X_n \) where \( Table[ele] \neq 0 \, \forall \, ele \in elements \)
5: \hspace{1em} Sort elements in descending order
6: \hspace{1em} bucketIndex = 1
7: \hspace{1em} for ele in elements do
8: \hspace{2em} Add ele in all_buckets[bucketIndex]
9: \hspace{2em} bucketIndex++
10: end for
11: end function

---

For a given \( n \) and \( S \), time complexity of the algorithm depends on the maximum number of subsets and time to find a bucket for each element placement. Since, finding the bucket is an iterative algorithm, time taken for this sub-method is also proportional to the number of subsets, \( SD[n][S] \). Since, the value of maximum number of subsets has exponential bound, \( O(2^n \times n^2) \), as described in Appendix 7, time
Algorithm 5 Basic BA: FindBucket(all_buckets, ele, S)

1: function FindBucket(all_buckets, ele, S)
2:     for bucket in all_buckets do
3:         if any bucket entry is same as ele then
4:             Next
5:         else if on adding ele in bucket, Sum(bucket) > S then
6:             Next
7:         else if on adding ele in bucket, bucket becomes duplicate to any other bucket then
8:             Next
9:         else
10:             Return bucket
11:         end if
12:     end for
13:     Return False
14: end function

Algorithm 6 Basic BA: Generating Subsets(n, S, SD[n][S], ED[n][S], prevWrongSubsets)

1: Given: n, S, SD[n][S] and ED[n][S][i] where i ∈ [1, n]
2: desiredSubsets = [] ▷ desiredSubsets are all the subsets of Xn with sum S.
3: wrongSubsets = [] ▷ wrongSubsets are the set of undesired and smaller subsets.
4: Table = GetED(n, S, ED[n][S], prevWrongSubsets) ▷ the count of every element in subsets of Xn with sum S
called from function GetED described in Algorithm 3
5: p = SD[n][S] : number of subsets of Xn with sum S

1: function GenerateSubsets
2:     all_buckets = p empty buckets
3:     initializeBuckets(all_buckets, Table, n, S, p) ▷ Initial Step
4:     fillBuckets = True ▷ Flag to control implementation of the while loop
5:     while (fillBuckets is set & (|all_buckets| > 0)) do
6:         fillBuckets = False
7:         q = count of non-zero entries of Table
8:         x = min(p, q)
9:         elements = x largest integers of Xn where Table[ele] ≠ 0 ∀ ele ∈ elements
10:        Sort elements in descending order
11:        for ele ∈ elements do
12:         if b is a bucket then ▷ When an element can be inserted in a valid bucket.
13:             Add ele in bucket b
14:             fillBuckets = True ▷ If no element is allotted to any bucket in a full iteration.
15:             Table[ele] --
16:         end if
17:         if Sum of the bucket b == S then
18:             desiredSubsets+ = b
19:             print bucket b
20:             Remove b from all_buckets
21:         end if
22:     end for
23:     end while
24:     for bucket ∈ remaining_buckets do
25:         wrongSubsets+ = bucket
26:     end for
27:     Return wrongSubsets, Table
28: end function
AlGORITHM 7 Basic BA: main Function(n, S)

1: function MAINFUNCTION(n, S)
2: prevWrongSubsets = |
3: prevTable = ED[n][S]
4: countSubsets = SD[n][S]
5: while countSubsets > 0 do
6: prevWrongSubsets, prevTable = GeneratingSubsets(n, S, countSubsets,
7: prevTable, prevWrongSubsets)
8: countSubsets = SD[n][S] = |prevWrongSubsets|
9: return True
10: end function

complexity is \( O(\text{max}(SD[n][S]) \cdot \text{max}(SD[n][S])) = O(2^n \cdot n^{\frac{2}{2}} \cdot 2^n \cdot n^{\frac{2}{2}}) = O(2^{2n} \cdot n^{-3}) \). Therefore, given \( n \) and \( S \), the time complexity to generate all the subsets is \( O(2^{2n} \cdot n^{-3}) \). Space complexity includes size of two storages Table and all buckets, \( O(n) + O(2^n) = O(2^n) \).

5.5 Subset Generation using Frequency Driven Bucket Algorithms

After the basic bucket algorithm we present two more bucket algorithms. While the previous algorithm uses the direct information provided by element distribution, \( ED[n][S][e] \) and sum distribution, \( SD[n][S][e] \), in these two algorithms we use element distribution in decreasing or increasing order. In other words, instead of assigning elements to a corresponding bucket in descending order, we assign elements to buckets based on their frequencies. Frequency of an element in all the subsets of \( X_n \) with sum \( S \), by definition, is equal to the count of the elements, denoted by \( ED[n][S][e] \). These algorithms are called Frequency-Driven (FD) Bucket algorithms. These can be called minimum FD or maximum FD bucket algorithms.

Information used by these algorithms is same as the Basic Bucket Algorithm. While the basic bucket algorithm is iterative, the minimum or maximum frequency driven algorithms are recursive. Information required by this algorithm, properties of elements that should be followed and the measures by which we ensure uniqueness (i.e. using log and lookup tables) is same as the primitive algorithm defined in Section 5.4.

Next, we generate all twenty subsets of \( X_{10} \) with \( \text{Sum} = 15 \). For both the algorithms, we select an element based on minimum or maximum frequency. In case of Minimum FD bucket algorithm, we select the maximum element with minimum frequency and recursively produce all the subsets of desired sum. For Maximum FD, we select maximum element with maximum frequency. In Table 6, we log all the iterations for generating all twenty subsets of \( X_{10} \) with \( \text{Sum} = 15 \). Following points briefly describe the working of Minimum FD bucket algorithm:

1. By following the algorithm, we select element 10. Since, \( ED[10][15][10] = 3 \), first iteration generates 3 subsets: \{\{10, 5\}, \{10, 4, 1\}, \{10, 3, 2\}\}. This is shown in the first row of Table 7. All subsets are generated in eight iterations.
2. In next three iterations, we choose elements-9, 8 and 7 respectively, to generate next thirteen subsets. This will results in production of sixteen subsets.
3. In every iteration we update the count of elements according to the resulting subsets.
4. In fifth iteration, we select element 2 and recursively generate two subsets, \{\{2, 6, 4, 3\}, \{2, 5, 4, 3, 1\}\}.

For maximum frequency driven bucket algorithm we select the maximum element with maximum frequency in every iteration.

1. All twenty desired subsets are produced in seven iterations.
2. Since, \( ED[10][15][1] = ED[10][15][2] = 9 \) and \( \text{max}(1, 2) \), we select element 2 and generate nine subsets.
3. In second iteration we select element 4 and recursively generate next four subsets: \{\{9, 6\}, \{9, 5, 1\}, \{9, 4, 2\} and \{9, 3, 2, 1\}\}.
4. Table 6 and Table 7 presents the log entries and subsets corresponding to all iterations of maximum FD bucket algorithm for \( X_{10} \) with \( \text{sum} = 15 \).
Table 6: Log table for iterations of Minimum and Maximum Frequency Driven Bucket Algorithm. We are generating all twenty subsets of $X_{10}$ with Sum = 15. Every column denotes the frequency calculation for ten elements and every row denotes the frequency calculations in every iteration. In this table the frequency of every selected element in the previous iteration is marked as bold.

| Iterations | Selected Element | Subsets                                      | [Subsets] |
|------------|------------------|----------------------------------------------|-----------|
| 1st iteration | 10               | $\{\{10,5\}, \{10,4,1\}, \{10,3,2\}\}$ | 3         |
| 2nd iteration | 9                | $\{\{9,6\}, \{9,5,1\}, \{9,4,2\}, \{9,3,2,1\}\}$ | 4         |
| 3rd iteration | 8                | $\{\{8,7\}, \{8,6,1\}, \{8,5,2\}, \{8,3,4\}, \{8,4,2,1\}\}$ | 5         |
| 4th iteration | 7                | $\{\{7,6,2\}, \{7,5,3\}, \{7,5,2,1\}, \{7,4,3,1\}\}$ | 4         |
| 5th iteration | 2                | $\{\{2,6,4,3\}, \{2,5,4,3,1\}\}$          | 2         |
| 6th iteration | 4                | $\{\{4,6,5\}\}$                           | 1         |
| 7th iteration | 6                | $\{\{6,5,3,1\}\}$                         | 1         |
| Total Number of Subsets |                    |                                              | 20        |

Table 7: Log table for iterations of Minimum and Maximum Frequency Driven Bucket Algorithm. We are generating all twenty subsets of $X_{10}$ with Sum = 15. First column represent all the iterations, second column shows the chosen element as per the frequency. Third column stores the subsets and the fourth column denotes the count of these subsets.

| Iterations | Selected Element | Subsets                                      | [Subsets] |
|------------|------------------|----------------------------------------------|-----------|
| 1st iteration | 2                | $\{\{2,1,3,4,5\}, \{2,1,3,9\}, \{2,1,4,8\}, \{2,1,5,7\}, \{2,3,4,6\}, \{2,3,10\}, \{2,4,9\}, \{2,5,8\}, \{2,6,7\}\}$ | 9         |
| 2nd iteration | 4                | $\{\{4,1,3,7\}, \{4,1,10\}, \{4,3,8\}, \{4,5,6\}\}$ | 4         |
| 3rd iteration | 1                | $\{\{1,5,9\}, \{1,6,8\}, \{1,3,5,6\}\}$ | 3         |
| 4th iteration | 10               | $\{\{10,5\}\}$                           | 1         |
| 5th iteration | 9                | $\{\{9,6\}\}$                           | 1         |
| 6th iteration | 8                | $\{\{8,7\}\}$                           | 1         |
| 7th iteration | 7                | $\{\{7,5,3\}\}$                         | 1         |
| Total Number of Subsets |                    |                                              | 20        |
5.6 Algorithm and Complexities

We state the pseudo codes for solving minimum and maximum FD bucket algorithms. Algorithm 8 updates the element distribution after every iteration and is called from Algorithm 10. This update ensures that correct number of subsets are generated. In Line 5, the element count is reduced according to the answer generated so far. The main function which was defined in Algorithm 9 repeatedly calls the `GeneratingSubsetsbyFDBucketAlgo` function and updates following information:

1. `countSubsets` - No. of subsets left.
2. `fullTable` - Element distribution of $X_n$ with $Sum = S$.
3. `elements` - Remaining elements which form remaining subsets.
4. `elementsCovered` - Elements which are not allowed or required to form remaining subsets.

Apart from these helper methods, the main functionality is presented in Algorithm 10. First we define the input for our algorithm. Line 2 is the base case of our recursive algorithm. We terminate the recursion when the desired sum $S$ is less than zero or there are no elements left to generate the subsets. In Line 7 and Line 8 we find $minKey$ and $minVal$ pair. In case of Minimum FD algorithm ($minKey, minVal$) is the largest element with minimum frequency, $e \in [1, n]$ where $ED[n][S][e]$ is minimum. For maximum FD algorithm, we find ($maxKey, maxVal$), the largest element with maximum frequency. The pseudo code for both algorithms are similar. Therefore, only minimum FD bucket algorithm is described. The main idea behind this algorithm is to find $minKey$, and generate subsets of $X_n$ with $Sum = [S−minKey]$. This means by adding $minKey$ to `elementsCovered`, in Line 10, we do not include it in future partial subsets. In Line 11, we recursively call `GeneratingSubsetsbyFDBucketAlgo` function with modified values. The remaining part of the code is divided in two conditions which are based on the return values from Line 11. It can either be empty or non-empty. $minKey$ is appended to every returning subset of $desiredSubsets[S−minKey]$ and `elementsCovered` are updated accordingly. In last few lines, we increase the count of $ED[n][S][e]$ for next iteration. This step ensures that the correct subsets are created in next iteration too.

For a given $n$ and $S$, time complexity of the maximum or minimum FD bucket algorithm depends on the maximum number of subsets and time taken to solve one recursion. Since, iterating through all elements is a recursive algorithm, time taken for this sub-method is also proportional to the number of subsets, $SD[n][S]$. Since, the value of maximum number of subsets has exponential bound, $O(2^n \cdot n^{\frac{2}{3}})$, as described in Appendix 4, time complexity is $O(max(SD[n][S] \cdot max(SD[n][S])) = O(2^n \cdot n^{\frac{2}{3}} \cdot 2^n \cdot n^{\frac{2}{3}}) = O(2^{2n} \cdot n^{-3})$. Therefore, for given $n$ and $S$, the time complexity to generate all the subsets is $O(2^{2n} \cdot n^{-3})$. Space complexity includes size of two storages `Table` and `desiredSubsets`, $O(n) + O(2^n) = O(n)$.

Algorithm 8 Max FD: GetED($n, S, Table, desiredSubsets$)

1: function GetED($n, S, Table, desiredSubsets$)
2:   newTable = Table
3:   for subset $\in$ desiredSubsets do
4:     for ele $\in$ subset do
5:       newTable[ele] = 1 $\triangleright$ Reducing count of elements according to desiredSubsets.
6:     end for
7:   end for
8:   Return newTable
9: end function

5.7 Subset Generation using Local Search

Our next enumeration technique for subset generation is called the Local Search. Before proceeding with this algorithm, we define two new types of subsets called Maximal and Minimal subsets. They act as the starting point for the local search algorithm.
Maximal and Minimal Subsets We present a new idea to categorize subsets of a given class. First, we divide the power set of $X_n$, $P(X_n)$, on the basis of their sum and then further partition these subsets according to their length. We have formulated and explained this selection process in Section 4.2.

For defining maximal subset we have the set of first $n$ natural numbers, $X_n$, sum$(S)$ which belongs to $[0, \text{maxSum}(n)]$ where $\text{maxSum}(n) = \frac{n(n+1)}{2}$ and length($l$) which belongs to $[0, n]$. Consider, $A$ denotes the subsets of $X_n$ with length $l$ and sum $S$. We denotes $A$ as $A = \{A_1, A_2 \ldots A_k\}$ where $k = LD[n][S][l]$, the total count of subsets with length $l$ and sum $S$. $A_i$ represents $i^{th}$ subset of set $A$ and $A_{l,j}$ represents $j^{th}$ element of subset $A_i$ where $j \in [1,l]$. There exists a maximal subset of $X_n$ of length $l$ and sum $S$, $A_{\text{maximal}}$, is defined such that $\forall j \in [1,l], A_{\text{maximal},j} > A_{p,j}$ where $p \in \{[1,k] - \text{maximal}\}$. There also exists a minimal subset, $A_{\text{minimal}}$, defined such that $\forall j \in [1,l], A_{\text{minimal},j} > A_{p,j}$ where $p \in \{[1,k] - \text{minimal}\}$.

The key point is that not all values of $A_{\text{maximal},j}$ will be greater than $j^{th}$ element of other subsets in $A$ but there will surely be a subset for which first $q$ elements are greater than rest of the subsets, where $q \in [1,l]$. In order to generate the subset $A_{\text{maximal}}$ for $X_n$ for a given sum $S$ and length $l$, we find the smallest possible element for every position, starting from the rightmost position. This pattern of element generation will ensure largest possible elements at the start of the subset, resulting in the maximal subset. Similarly, we find the largest possible element for every position of minimal subset starting from the rightmost position which ensures the smallest possible element at the start of the subset, resulting in the desired minimal subset. Table 8 displays the maximal and minimal subsets for every sum and length pair of $X_5$.

The core idea for the local search algorithm is to find all possible subsets of a particular length $l$ and sum $S$ where our starting subset can be a maximal or minimal subset. We find subsets by iterating over length between $l_{\text{min}}$ and $l_{\text{max}}$ where these are the minimum and maximum possible subsets of $X_n$ with sum $s$ respectively. This is a heuristic algorithm. Next, we present a few examples to explain local search using maximal and minimal subset respectively.

Maximal subset has the largest possible element at every position for a given sum $S$ and length $l$. Therefore, for local search starting with the maximal subset, we begin from left most element, decrement the first permissible element followed by increment of next permissible element. On contrary, minimal subset has smallest possible element at every position for a given sum $S$ and length $l$. Therefore, we begin from left most element, increment the first permissible element followed by decrement of next permissible element. Every increment or decrement consists of one unit.

1. Figure 1 shows the local search example for $n = 10$, sum = 21 and length = 3 where the starting subset is the maximal subset of respective length.

(a) We start with subset $\{6, 7, 8\}$. By decrementing the first permissible element 6 by 1 and incrementing third permissible element 8 by 1, we generate the second subset $\{5, 7, 9\}$. We cannot increment the second element of subset $\{6, 7, 8\}$, as on incrementing 7 by 1, we get 8 which creates duplication. In this case, 7 is a non-permissible element.

(b) Next, we generate subsets :$\{4, 8, 9\}, \{5, 6, 10\}, \{4, 7, 10\}$ from subset $\{5, 7, 9\}$.

(c) By following the same procedure, we generate all desired subsets of $X_{10}$ with sum 21 and length 3 from a single maximal set $A_{\text{maximal}}$.  

---

### Algorithm 9 Max FD: main Function($n$, $S$)

1: function mainFunction($n$, $S$)  
2:   fullTable = $ED[n][S]$  
3:   countSubsets = $SD[n][S]$  
4:   elements = $[1, 2 \ldots n]$  
5:   elementsCovered = $\{\}$  
6:   while countSubsets > 0 do  
7:      desiredSubsets = GeneratingSubsets($n$, $S$, countSubsets, elements, elementsCovered, fullTable)  
8:      countSubsets = $SD[n][S] - |\text{desiredSubsets}|$  
9:      Update fullTable, elements, elementsCovered  
10: end while  
11: Return True  
12: end function
Algorithm 10 Max FD: GeneratingSubsets by FDBucketAlgo

1: Given: \( n, S, SD[n][S] \) and \( ED[n][S][i] \) where \( i \in [1, n] \)
2: \( \text{desiredSubsets} = [ ] \) \( \triangleright \) \( \text{desiredSubsets} \) are all the subsets of \( X_n \) with sum \( S \).
3: \( \text{fullTable} = \text{GetED}(n, S, ED[n][S], \text{desiredSubsets}) \) \( \triangleright \) the count of every element in subsets of \( X_n \) with sum \( S \)
called from function \( \text{GetED} \) described in Algorithm 8
4: \( p = SD[n][S] \) : number of subsets of \( X_n \) with sum \( S \)

1: \( \text{function GenerateSubsets} \)
2: \  \( \text{if } S \leq 0 \) or \( |\text{elements}| == 0 \) then
3:     \( \text{Return desiredSubsets} \)
4: \  \( \text{end if} \)
5: \  \( \text{countSubsets} = SD[n] \)
6: \  \( \text{while countSubsets} > 0 \) do
7:     \( \text{minVal} = \min(ED[n][S][e] > 0) \)
8:     \( \text{minKey} = \max(e \in [1, n] \& ED[n][S][e] == \text{minVal}) \)
9:     \( \text{elements.remove(minKey)} \)
10: \  \( \text{elementsCovered.add(minKey)} \)
11: \  \( \text{desiredSubsets} = \text{GeneratingSubsets}(n, S - \text{minKey}, \text{countSubsets} \), \text{elements}, \text{elementsCovered}, \text{fullTable}) \)
12: \  \( \text{if desiredSubsets}[S - \text{minKey}] \) is empty then
13:     \( \text{countSubsets} -- \)
14:     \( ED[n][S][\text{minKey}] -- \)
15:     \( \text{desiredSubsets}[S] = [[\text{minKey}]] \)
16:     \( \text{print(desiredSubsets}[S]) \)
17:     \( \text{elementsCovered.remove(minKey)} \)
18: \  \( \text{else} \)
19:     \( \text{for } A \in \text{desiredSubsets}[S - \text{minKey}] \) do
20:         \( ED[n][S][\text{minKey}] -- \)
21:         \( \text{if (minKey} \notin A \) \& (minKey + \sum(A) == S) \) then
22:             \( \text{if A.append(minKey) is unique then} \)
23:                 \( \text{countSubsets} -- \)
24:                 \( \text{print(A)} \)
25:                 \( \text{desiredSubsets}[S].append(A) \)
26:                 \( \text{In elementsCovered add elements of } A \)
27:             \( \text{end if} \)
28:         \( \text{end if} \)
29:     \( \text{end for} \)
30: \  \( \text{end if} \)
31: \  \( \text{for } e \in ED[n][S][e] <= 0 \) do
32:     \( \text{elementsCovered.add(e)} \)
33: \  \( \text{end for} \)
34: \  \( \text{for } A \in \text{desiredSubsets} \& e \in A \) do
35:         \( ED[n][S][e] + + \)
36: \  \( \text{end for} \)
37: \  \( \text{Return desiredSubsets} \)
38: \text{end function}
| Sum | Length | Subsets | Maximal Subset | Minimal Subset |
|-----|--------|---------|----------------|---------------|
| 0   | 0      | $\phi$  | $\phi$         | $\phi$        |
| 1   | 1      | $\{1\}$ | (1)            | (1)           |
| 2   | 1      | $\{2\}$ | (2)            | (2)           |
| 3   | 1      | $\{3\}$ | (3)            | (3)           |
|     | 2      | $\{1,2\}$ | $\{1,2\}$     | $\{1,2\}$    |
| 4   | 1      | $\{4\}$ | (4)            | (4)           |
|     | 2      | $\{1,3\}$ | $\{1,3\}$     | $\{1,3\}$    |
| 5   | 1      | $\{5\}$ | (5)            | (5)           |
|     | 2      | $\{2,3\}, \{1,4\}$ | (2,3)         | (1,4)         |
| 6   | 2      | $\{2,4\}, \{1,5\}$ | (2,4)         | (1,5)         |
|     | 3      | $\{1,2,3\}$ | (1,2,3)       | (1,2,3)       |
| 7   | 2      | $\{3,4\}, \{2,5\}$ | (3,4)         | (2,5)         |
|     | 3      | $\{1,2,4\}$ | (1,2,4)       | (1,2,4)       |
| 8   | 2      | $\{3,5\}$ | (3,5)         | (3,5)         |
|     | 3      | $\{1,3,4\}, \{1,2,5\}$ | (1,3,4)       | (1,2,5)       |
| 9   | 2      | $\{4,5\}$ | (4,5)         | (4,5)         |
|     | 3      | $\{2,3,4\}, \{1,3,5\}$ | (2,3,4)       | (1,3,5)       |
| 10  | 3      | $\{2,3,5\}, \{1,4,5\}$ | (2,3,5)       | (1,4,5)       |
|     | 4      | $\{1,2,3,4\}$ | (1,2,3,4)     | (1,2,3,4)     |
| 11  | 3      | $\{2,4,5\}$ | (2,4,5)       | (2,4,5)       |
|     | 4      | $\{1,2,3,5\}$ | (1,2,3,5)     | (1,2,3,5)     |
| 12  | 3      | $\{3,4,5\}$ | (3,4,5)       | (3,4,5)       |
|     | 4      | $\{1,2,4,5\}$ | (1,2,4,5)     | (1,2,4,5)     |
| 13  | 4      | $\{1,3,4,5\}$ | (1,3,4,5)     | (1,3,4,5)     |
| 14  | 4      | $\{2,3,4,5\}$ | (2,3,4,5)     | (2,3,4,5)     |
| 15  | 5      | $\{1,2,3,4,5\}$ | (1,2,3,4,5)   | (1,2,3,4,5)   |

Table 8: Maximal and minimal subsets for every sum and length pair of $X_5$. 
2. Figure 2 presents the local search example for $n = 10$, sum = 21 and length = 3 where the starting subset is the minimal subset of respective length.
   (a) We start with subset $\{2, 9, 10\}$. By incrementing the first permissible element 2 by 1 and decrementing the second permissible element 9 by 1, we generate the second subset $\{3, 8, 10\}$. We can not decrement the third element of subset $\{2, 9, 10\}$, as on decreasing 10 by 1, we get 9 which leads to duplication. In this case, 10 is a non-permissible element.
   (b) Next, we generate subsets : $\{\{4, 7, 10\}, \{4, 8, 9\}\}$ from subset $\{3, 8, 10\}$.
   (c) By following the same procedure, we generate all desired subsets of $X_{10}$ with sum 21 and length 3 from a single minimal set, $A_{\text{minimal}}$.

3. While generating a subset using Local Search Algorithm, we ensure that the sum of subset is equal to the desired target sum $S$, the subset do not contain duplicates and there is uniqueness among the subsets. Uniqueness among and within these subset is ensured by using lookup technique introduced in Section 7. This establishes the correctness of the Local Search Algorithms using Maximal and Minimal Subsets.

4. Since we know the count of all subsets of $X_n$ with $\text{Sum} = S$ and $\text{Length} = l$, we generate all the subsets and this approach is concluded only when all desired subset results are achieved. This establish the completeness of the Local Search Algorithms using Maximal and Minimal Subsets.

---

**Local Search using Maximal Subset:** Algorithm 11 presents a procedure to generate all subsets of $X_n$ with particular sum $S$ and length $l$ where the seed subset is the maximal subset, $A_{\text{maximal}}$. We begin from the left most element, decrement the first permissible element followed by increment of next permissible element. Each increment or decrement consists of one unit. In Algorithm 11 we use a queue data structure to
store all the resulting subsets, including maximal subset. We can iterate all the subsets in FCFS manner via these method. We check the uniqueness among the subsets by using the concept of lookup table as defined in Section 7. A subset is pushed in the queue only if its unique. This algorithm is terminated when all the subsets are generated.

**Local Search using Minimal Subset:** Algorithm 12 represents a procedure to generate all subsets of $X_n$ with particular sum $S$ and length $l$ where the seed subset is the minimal subset, $A_{\text{minimal}}$. We begin from left most element, increment the first permissible element followed by decrement of next permissible element. Every increment or decrement consists of one unit. Algorithm 12 uses the same queue data structure and checks the uniqueness among the subsets by using the concept of lookup table as Algorithm 11. This algorithm is terminated when all subsets are generated.

**Complexities:** Time complexity of these algorithms is complexity of while loop $\times$ complexity of for loop, i.e., maximum no. of subsets $\times$ length of each subset. The complexity of the length of each subset variable is $O(n)$ but the time complexity of maximum no. of subsets variable is exponential. This makes the algorithm exhaustive. Time complexity is $O(2^{\frac{n}{2}} \cdot n) = O(2^{\frac{n}{2}}) = O(\frac{2^n}{\sqrt{n}})$. The space complexity for these algorithms is equal to the size of storage queue i.e. maximum no. of subsets $\times$ length of each subset. The time complexity is similar. The complexity of the Length of each subset variable is $O(n)$ but the space complexity of maximum no. of subsets variable is exponential, $O(\frac{2^n}{\sqrt{n}})$.

**Algorithm 11 LS MaxS: Local Search for Maximal Subset**

```plaintext
1: function LOCALSEARCH(n, s, l)
2:   maximalSet = MAXIMALSUBSET(n, s, l)
3:   queue.push(maximalSet)
4:   allSubsetsGenerated = LD[n][s][l]
5:   while allSubsetsGenerated > 0 do
6:     reqSet = queue.pop()
7:     for i = 1; i ≤ len - 1; i++ do
8:       if reqSet[i] - 1 > reqSet[i - 1] then
9:         reqSet[i] = 1  ▷ First decrementing the element by 1
10:        decrement = True
11:     end if
12:     for j = i + 1; j ≤ l; j++ do
13:       if reqSet[j] + 1 < reqSet[j + 1] then
14:         reqSet[j] = 1
15:         increment = True
16:       end if
17:     end if
18:     if (reqSet is unique) and (decrement) and (increment) then
19:       print reqSet
20:       queue.push(reqSet)
21:       allSubsetsGenerated --
22:     end if
23:   end while
24: end function
```

6 Experimental Results

This section presents the experiments that we have conducted to validate the efficiency and effectiveness of all the proposed algorithms.
Algorithm 12 LS MinS: Local Search for Minimal Subset

1: function localSearch(n, s, l)
2: minimalSet = minimalSubset(n, s, l)
3: queue.push(minimalSet)
4: allSubsetsGenerated = LD[n][l][s]
5: while allSubsetsGenerated > 0 do
6: reqSet = queue.pop()
7: for i = 1; i ≤ len − 1; i + + do
8: if reqSet[i] + 1 < reqSet[i + 1] then
9: reqSet[i] + = 1 ▷ First incrementing the element by 1
10: increment = True
11: end if
12: for j = i + 1; j ≤ l; j + + do
13: if reqSet[j] − 1 > reqSet[j − 1] then
14: reqSet[j] − = 1
15: decrement = True
16: end if
17: if (reqSet is unique) and (decrement) and (increment) then
18: print reqSet
19: queue.push(reqSet)
20: allSubsetsGenerated −−
21: end if
22: end for
23: end for
24: end while
25: end function

6.1 Summary of Alternate Enumeration Techniques

Following table summarizes all the alternate enumeration techniques to solve SSP.

| Problem Statement | Algorithm | Core Idea | Time Complexity | Space Complexity |
|-------------------|-----------|-----------|-----------------|-----------------|
| Find all subsets of \( P(X_n) \) which sum up to \( S \), where \( X_n \) is the set of first \( n \) natural numbers, \( X_n = \{1, 2, \ldots , n\} \) | Backtracking Algorithm (Naive) (section 5.1) | It is an improved and systematic brute force approach for generating various subsets with \( \text{Sum} = S \). We iterate through all \( 2^n \) solutions in an orderly fashion. | \( \mathcal{O}(n \times 2^n) \) | \( \mathcal{O}(n) \) |
| Subset Generator using Sum Distribution (SDG) (section 5.2) | This algorithm is a recursive generator based on the concept of Sum Distribution and uses subsets of \( X_{(n−1)} \) to produce results for \( X_n \). Subsets of \( X_n \) with \( \text{Sum} = S \) are generated by subsets of \( X_{n−1} \) with \( \text{Sum} = (S−n) \). | \( \mathcal{O}(2^n \times n^2) \) | \( \mathcal{O}(2^n \times n^2) \) |
| Subset Generator using Length-Sum Distribution (LDG) (section 5.3) | This algorithm is a recursive generator based on the concept of Length-Sum Distribution and uses subsets of \( X_{(n−1)} \) to produce results for \( X_n \). Subsets of \( X_n \) with \( (\text{Sum} = S, \text{Length} = l) \) are generated by subsets of \( X_{n−1} \) with \( (\text{Sum} = S−n, \text{Length} = l−1) \). | \( \mathcal{O}(2^n \times n^2) \) | \( \mathcal{O}(2^n \times n^2) \) |
| Algorithm Name | Core Idea | Time Complexity |
|----------------|-----------|-----------------|
| Basic Bucket Algorithm (Basic BA) | The basic idea behind this enumeration technique is to use the various distribution values. We consider $SD[n][S]$ number of empty buckets, storage data structures, and iterate through all elements in descending order. During each iteration an element is assigned to one of the buckets. This method is about adding the correct element to the corresponding subset. This is an iterative algorithm. | $O(2^n \cdot n^{-3})$ |
| Maximum Frequency Driven Bucket Algorithm (Max FD) | Information used by this recursive algorithm is same as the basic bucket algorithm. Instead of choosing elements in descending order, we select maximum element with maximum frequency to generate all $SD[n][S]$ number of subsets of $X_n$ with $Sum = S$. | $O(2^n \cdot n^{-3})$ |
| Minimum Frequency Driven Bucket Algorithm (Min FD) | This algorithm is contrary to maximum FD bucket algorithm. Information used by this is also similar to the basic bucket algorithm. We select maximum element with minimum frequency to generate all $SD[n][S]$ number of subsets of $X_n$ with $Sum = S$. | $O(2^n \cdot n^{-3})$ |
| Local Search using Maximal Subset (LS MaxS) | This heuristic algorithm finds all desired subsets by choosing the maximal subset as the seed. Maximal subset has the largest possible element at every position for a given sum($S$) and length($l$). Therefore, we begin from left most element, decrement the first permissible element followed by increment of next permissible element. Every increment or decrement consists of one unit. | $O\left(\frac{2^n}{\sqrt{n}}\right)$ |
| Local Search using Minimal Subset (LS MinS) | This heuristic algorithm also finds all desired subsets by choosing the minimal subset as the seed (starting point). Minimal subset has smallest possible element at every position for a given sum($S$) and length($l$). Therefore, we begin from left most element, increment the first permissible element followed by decremental of next permissible element. Every increment or decrement consists of one unit. | $O\left(\frac{2^n}{\sqrt{n}}\right)$ |

Table 9: Summary of the core concepts and ideas of all the alternate enumeration techniques to solve subset sum problem. First column introduces every algorithm, second column presents the core idea behind the algorithm and the last two columns states their time and space complexities.

6.2 Experimental Setup

We have carried out various sets of experiments on an i7-2600 machine with 64GB of RAM to compare and analyze the performance of our algorithms under various considerations. We define the experimental setup and measuring parameters before comparing the performances.

Due to symmetric property of SSP, we choose random sum values in lower part of the sum range i.e. $S \leq midSum(n)$. In Figure 3, we show different plots for number of subsets of $X_n$ for various sums. These figures help us estimate the problem space for generating results of alternate enumeration techniques. We select the value of $S$ as $2n$ to show the behavior of number of subsets of $X_n$ with sum $S$ when $S$ has the complexity $O(n)$. Similarly, we choose the value of $S$ as $midSum(n) - n$ because the number of subsets of $X_n$ with this sum are in order of $O(midSum(n))$. This upper bound of the Sum Distribution $SD[n][midSum(n)] = S(n) \approx \sqrt{\frac{6}{\pi}} \cdot 2^n \cdot n^{\frac{n}{2}}$ is explained in Appendix 7. Table 10 presents the count of number of subsets $X_n$ with $S = 2n$ and $S = \left(\frac{n(n+1)}{4}\right) - n$. This table gives an estimate of the values...
plotted in Figure 3. Figure 3(a, c) plot the number of subsets of \( X_n \) with \( (n \in [1, 250], \, S = 2^n) \) and \( (n \in [1, 50], \, S = \left( \frac{n(n+1)}{4} - n \right)) \) respectively. Figure 3(b, d) plot the log to the base 10 of number of subsets of \( X_n \) with \( (n \in [1, 250], \, S = 2^n) \) and \( (n \in [1, 50], \, S = \left( \frac{n(n+1)}{4} - n \right)) \) respectively. Since the values of number of subsets for a particular \( S \) increases exponentially with \( n \), we have plotted Figure 3(b, d) by using the logarithmic function. This helps in approximating the size of the problem space.

| \( n \) | Count of Subsets of \( X_n \) with \( S = 2^n \) | \( n \) | Count of Subsets of \( X_n \) with \( S = \left( \frac{n(n+1)}{4} - n \right) \) |
|-------|-----------------|-------|----------------|
| 6     | 2               | 6     | 2               |
| 7     | 5               | 7     | 5               |
| 8     | 8               | 8     | 8               |
| 9     | 13              | 9     | 13              |
| 10    | 134             | 10    | 24              |
| 50    | 416868          | 15    | 521             |
| 100   | 482240364       | 20    | 11812           |
| 150   | 114613846376    | 30    | 7206286         |
| 200   | 11954655830925  | 40    | 5076120114      |
| 250   | 732839540340934 | 50    | 3831141038816   |

Table 10: Count of number of subsets \( X_n \) with \( S = 2^n \) and \( S = \left( \frac{n(n+1)}{4} - n \right) \) respectively.

### 6.3 Excess Subset Generation Analysis

Given \( X_n \) and a sum \( S \), we know how many subsets of \( X_n \) have sum \( S \). This value is \( SD[n][S] \). For each algorithm, in order to generate these \( SD[n][S] \) subsets we may explore few extra subsets of \( X_n \) whose sum not equal to \( S \).

In naive backtracking method, at every step of subset generation we either include or exclude an element. This creates a recursive tree and a branch is terminated when the current sum exceeds the target. This way we explore more subsets than desired sum. Similarly, in rest of the alternate enumeration techniques in order to generate all subsets of \( X_n \) with sum \( S \), we explore more subsets than desired number of subsets. In this analysis we measure this extra exploration. In Table 11 we present the ratios of subsets explored to total number of subsets of \( X_n \) with sum \( S \) i.e. \( SD[n][S] \). The first three columns of this table states \((n, S)\) pair and the value of \( SD[n][S] \) for all these pairs. The remaining eight columns denote the ratio of explored subsets to the number of subsets in the final solution for all eight alternate enumeration techniques. With every ratio we also represent the time taken by the algorithm to generate the solution set. For every value of \( n \) and \( S \), we bold the least ratio and least time taken by an algorithm.

Following observations can be made based on the data presented in Table 11.

1. For a given value of \( n \) and \( S \), desired ratio is a fraction of the number of subsets to be generated to the total number of subsets of \( X_n \) with Sum \( S \) i.e. \( SD[n][S] \). For example, \( n = 12 \) and \( S = 24 \), the number of subsets of \( X_{12} \) with Sum \( =24 \) are 67. Therefore, the value of \( SD[12][24] \) = 67.
2. Every column corresponding to a given algorithm presents the ratio of number of subsets explored to generate the desired subsets to the total number of subsets of \( X_n \) with sum \( S \). For example for naive algorithm, given \( n = 12 \) and \( S = 24 \), the number of subsets explored for generating all subsets of \( X_{12} \) with Sum \( =24 \) are 737. Therefore, the desired ratio for these values is: \( \frac{737}{67} \approx 11 \).
3. The ratio for all algorithms should be greater than the desired ratio. If not, then it implies that complete result has not been generated. In this table for a given \( n \) and \( S \), the ratio for all algorithms is greater
Fig. 3: Plot of number of subsets of $X_n$ against sums in smaller and larger ranges. For smaller range we select $S = 2n$ and plot graph for $n$ varying from 1 to 250. (a) Figure represents graph for number of subsets of $X_n$ with $S = 2n$ where $n \in [1, 250]$. (b) Figure represents graph for number of subsets of $X_n$ with $S = 2n$ with logarithmic base 10 where $n \in [1, 250]$. For larger range we select $S = \left(\frac{n(n+1)}{4} - n\right)$ and plot graph for $n$ varying from 1 to 50. (c) Figure represents graph for number of subsets of $X_n$ with $S = \left(\frac{n(n+1)}{4} - n\right)$ where $n \in [1, 50]$. (d) Figure represents graph for number of subsets of $X_n$ with $S = \left(\frac{n(n+1)}{4} - n\right)$ with logarithmic base 10 where $n \in [1, 50]$. 
than the desired ratio. This observation and the correctness of these algorithms ensure the completeness of the results.

4. Naive algorithm explores most number of subsets in order to generate the desired subsets. Naive is the worst performing enumeration technique compared to all our proposed algorithms. This shows that all our alternate enumeration techniques perform better than the benchmark algorithm.

5. Ratios of LS MaxS and LS MinS are closer to the desired ratio for a given $n$ and $S$.
   (a) Since LS MaxS and LS MinS are heuristic algorithms, they explore lesser number of subsets as compared to naive algorithm.
   (b) For example, given $n = 12$ and $S = 24$, the number of subsets explored for LS MaxS and LS MinS are 93 and 103 respectively creating a ratio of $\frac{93}{103} = 1.3881$ and $\frac{100}{67} = 1.5373$.
   (c) The drawback for these algorithm is that they do not generate results for higher values of $n$ and $S$ within short amount of time. This is explained more in Section 6.4

6. After Local Search algorithms, LDG and SDG are next in good performance ranking. Ratio for LDG is smaller and closer to desired ratio than SDG.
   (a) Since LDG is a simple dynamic algorithm which generate the subsets based on their sum and length, it goes to one more level of categorization among subsets and minimizes the excess exploration of undesired subsets.
   (b) Given $n = 12$ and $S = 24$, the number of subsets explored by LDG are 150 and ratio is $\frac{150}{97} = 2.3888$.
   (c) LDG has precedence over others as it can enumerate all subsets of $X_n$ for a considerable values of $n$ within short amount of time. The numbers are shown in Table 18 of Section 6.4.
   (d) SDG explores more subsets than LDG but it performs better than naive. While naive implementation involves a recursive tree based on the inclusion and exclusion of an element at every step, SDG builds the subset by using the exact formula defined in Section 4.
   (e) Given $n = 12$ and $S = 24$, the number of subsets explored by SDG are 214 and ratio is $\frac{214}{67} = 3.1940$.

7. Performance of Max FD and Min FD is similar to SDG. For $n = 12$ and $S = 24$, MaxFD explores 166 subsets and MinFD explores 241 subsets. For other pairs of $n$ and $S$ these values are very close.

8. Basic Bucket algorithm (Basic BA) also performs better than naive but can not compute all subsets for a considerable value of $n$ and $S$ within short amount of time.

| $n$ | $S$ | [Subsets] | Naive | SDG | LDG | Basic BA | Max FD | Min FD | LS MaxS | LS MinS |
|-----|-----|-----------|-------|-----|-----|----------|--------|--------|---------|---------|
| 12  | 24  | 67        | 11    | (0.00247) | 3.1940 | (0.009596) | 2.2388 | (0.00103) | 5.0896 | (1.618) | 2.4776 | (0.195) | 3.5970 | (1.822) | 1.3881 | (0.019) | 1.5373 | (0.016) |
| 12  | 27  | 84        | 20.5952 | (0.00178) | 2.7857 | (0.012512) | 2.0952 | (0.00116) | 5.1548 | (2.394) | 2.6190 | (0.405) | 4.2262 | (2.958) | 1.3690 | (0.02) | 1.2976 | (0.024) |
| 15  | 30  | 186       | 21.4194 | (0.00795) | 6.2097 | (0.014521) | 1.6882 | (0.00499) | 4.7742 | (21.208) | 2.3871 | (1.381) | 4 | (7.641) | 1.1882 | (0.11) | 1.2903 | (0.133) |
| 15  | 45  | 521       | 23.3282 | (0.01184) | 2.8177 | (0.056801) | 1.3013 | (0.00472) | - | (14.031) | 2.8503 | (13.53746) | 1.3129 | (353.746) | 1.0211 | (0.798) | 1.0058 | (0.955) |
| 16  | 32  | 253       | 60.6087 | (0.01118) | 8.2806 | (0.017711) | 1.6719 | (0.00615) | 5.4664 | (77.48) | 2.3478 | (2.341) | 3.7470 | (11.761) | 1.1621 | (0.211) | 1.2332 | (0.255) |
| 17  | 59  | 1764      | 27.0947 | (0.03996) | 2.9127 | (0.233748) | 1.3622 | (0.01556) | - | (153.31) | 2.9892 | (153.31) | - | (10.144) | 1.0176 | (12.058) |
| 20  | 40  | 860       | 73.6390 | (0.03277) | 30.0447 | (0.055301) | 1.8189 | (0.04656) | - | (21.923) | 2.2667 | (21.923) | - | (3.438) | 1.0707 | (2.81) | 1.1191 | (3.438) |
| 20  | 85  | 11812     | 28.4236 | (0.30453) | 2.9261 | (2.08991) | 1.6258 | (0.1357) | - | (698.574) | - | (698.574) | - | (572.839) | 1.2332 | (664.875) |
| 22  | 104 | 41552     | 34.7394 | (1.21103) | 2.9706 | (8.53939) | 1.8080 | (0.52493) | - | - | - | - | - | - | - | - | - |

Table 11: The ratios of subsets explored to total number of subsets of $X_n$ with sum $S$ i.e. $SD[n][S]$ is presented in this table. The first three columns of this table states ($n$, $S$) pair and the value of $SD[n][S]$ for all these pairs. The remaining eight columns denote subsets explored ratio for all eight alternate enumeration techniques. With every ratio we also represent the time taken by the algorithm to generate the solution set. The least ratio and least time taken for every $n$ and $S$ are presented in bold.
6.4 Comparative Analysis of Enumeration Algorithms

In this section, we present the time taken by various enumeration techniques under different conditions. Experiments defined in this section are categorized based on the range of input sum value corresponding to the set of natural numbers $X_n$. Given $X_n$, $\text{Sum}(A)$ belonging to the range $[0, \text{maxSum}(n)] = [0, \frac{n(n+1)}{2}]$ where $A \in P(X_n)$. Choosing different values of sum between 0 to $\text{maxSum}(n)$ is the core idea behind these experiments. Table 12 summarizes the explanation of all these experiments.

| Experiments / Comparative Analysis | Description and Examples | Algorithms | Tables or Figures |
|-----------------------------------|--------------------------|------------|------------------|
| CA-SSR $[1, 2n]$                 | For this experiment we randomly choose sum $S_1$ from a smaller range and calculate the time taken to generate subsets of $X_n$ with $\text{Sum} = S_1$. For every values of $n$, this smaller range varies from 1 to $2n$ i.e. $\forall n, S_1 \in [1, 2n]$. | Basic BA, Max FD, Min FD, LS MaxS, LS MinS | Table 13: Time taken (in seconds) by Basic BA, Max FD and Min FD in CA-SSR. Table 14: Time taken (in seconds) by LS MaxS and LS MinS in CA-SSR. |
| CA-LSR $[\text{midSum}(n) - n, \text{midSum}(n)]$ | For this experiment we randomly choose sum $S_2$ from a larger range and calculate the time taken by all the algorithms to generate subsets of $X_n$ with $\text{Sum} = S_2$. For every values of $n$, this larger range varies from $\text{midSum}(n) - n$ to $\text{midSum}(n)$ i.e. $\forall n S_2 \in [\text{midSum}(n) - n, \text{midSum}(n)]$. | Basic BA, Max FD, Min FD, LS MaxS, LS MinS | Table 15: Time taken (in seconds) by Basic BA, Max FD and Min FD in CA-LSR. Table 16: Time taken (in seconds) by LS MaxS and LS MinS in CA-LSR. |
| CA-FSV                           | In this experiment instead of choosing random values of $S$ for every algorithm against every $n$, we fix few pairs of $(n, S_1)$ and $(n, S_2)$ for all the algorithms where $S_1 = 2 \times n$ and $S_2 = \text{midSum}(n) - n$ | Basic BA, Max FD, Min FD, LS MaxS, LS MinS, LDG, SDG | Table 17 presents the time taken by Max FD, Min FD, Basic BA, LS MaxS, LS MinS, LDG and SDG algorithms where $S_1 = 2 \times n$ and $S_2 = \text{midSum}(n) - n$ |
| CA-SLN                           | For this experiment instead of fixing the value of sum $S$, we vary $S$ from 0 to $\text{maxSum}(n) = \frac{n(n+1)}{2}$. This experiment helps us in analyzing the performance of SDG and LDG algorithms against Naive (backtracking) algorithm. In this experiment we enumerate all $2^n$ subsets of $X_n$ | SDG, LDG, Naive | Table 18 presents the comparison of SDG and LDG with naive backtracking algorithm. This table presents the time taken (in sec) while enumerating all $2^n$ subsets of $X_n$ for every value of sum $S$ in range $[0, \frac{n(n+1)}{2}]$. This is the time taken by these algorithms to enumerate each and every subset. Figure 4: Plot of SDG, LDG and Naive algorithm while enumerating all $2^n$ subsets of $X_n$ for every value of sum $S$ in range $[0, \frac{n(n+1)}{2}]$. |

Table 12: Summary of the experimental setup for Comparative Analysis of Enumeration Algorithms. First column states the name, second columns describes the experiment, third column lists the algorithms for which the experiment is carried out and the fourth column presents the tables and figures stating the time taken by different algorithms under several conditions.
We have drawn these tables and shown these times for demonstrative purposes. We have observed the following by running all the eight algorithms:

1. From comparative analysis of algorithms in smaller range (CA-SSR) we can see that Basic BA, Max FD, Min FD, LS MaxS and LS MinS generate subsets till $n$ equal to 22, 36, 36, 44 and 44 respectively and takes less than 35,000 seconds.
   - Since LS MaxS and LS MinS explores lesser number of extra subsets as shown in Section 6.3 it takes lesser amount of time than bucket algorithms.
   - Among these five algorithms, Basic BA explores maximum number of subsets, takes most time for execution and can generate results till smaller values of $n$.
2. Comparative analysis of algorithms in larger range (CA-LSR) follows similar pattern as CA-SSR. The value of sum selected in this range has higher value of $SD[n][S]$ which results in more execution time. LS MaxS and LS MinS perform the best in this experiment.
3. Comparative Analysis with Fixed Sum Values (CA-FSV) allows us to compare seven algorithms: Max FD, Min FD, Basic BA, LS MaxS, LS MinS, LDG and SDG for a fixed value of $n$ and $S$.
   - From this comparative study, we can see that LDG and SDG outperforms all the other algorithms. They can be executed till $n = 36$ and takes least amount of time.
   - Even though SDG and LDG explores more number of subsets, additional information required by these algorithms is much lesser than the additional information required by Local Search and Bucket Algorithms.
   - SDG and LDG does not require the values of $SD[n][S]$ and $ED[n][S][e]$ at every step of execution. They do not need to maintain the current state of algorithm. This reduces the execution time.
4. From comparative analysis of SDG, LDG and Naive (CA-SLN) we compare LDG, SDG with naive to show that our alternate enumeration techniques performs better than the existing algorithms. Using naive algorithm, we are not able to generate all the subset above $n$ equal to or greater than 24. This limits the execution. But LDG and SDG can easily be computed till $n = 34$ in less than 40 minutes.

These timings are implementation and machine dependent. The above results show that even though some algorithms explore fewer extra subsets but they take more time due to lack of efficient implementation, storage and memory constraint.
| Time taken (in sec) by LS MaxS in CA-SSR | Time taken (in sec) by LS MinS in CA-SSR |
|----------------------------------------|----------------------------------------|
| \( n \) | \( S \) | Time (in sec) | \( n \) | \( S \) | Time (in sec) | \( n \) | \( S \) | Time (in sec) | \( n \) | \( S \) | Time (in sec) | \( n \) | \( S \) | Time (in sec) | \( n \) | \( S \) | Time (in sec) | \( n \) | \( S \) | Time (in sec) | \( n \) | \( S \) | Time (in sec) |
| 3   | 1    | 0.00015 | 24   | 47   | 19.862 | 3   | 1    | 0.00020 | 24   | 47   | 21.2260 | 4   | 1    | 0.00019 | 25   | 19   | 0.01893 |
| 4   | 1    | 0.00012 | 25   | 36   | 2.0454 | 4   | 1    | 0.00020 | 26   | 52   | 77.8854 | 5   | 1    | 0.00022 | 27   | 22   | 0.05621 |
| 5   | 1    | 0.00015 | 26   | 10   | 0.0023 | 5   | 2    | 0.00073 | 28   | 10   | 0.00278 | 6   | 2    | 0.00022 | 27   | 22   | 0.05621 |
| 6   | 3    | 0.00024 | 27   | 17   | 0.0114 | 6   | 3    | 0.00077 | 28   | 17   | 0.0114  | 7   | 7    | 0.00073 | 28   | 10   | 0.00278 |
| 7   | 4    | 0.00023 | 28   | 5    | 0.002251 | 7   | 7    | 0.00077 | 28   | 17   | 0.0114  | 8   | 6    | 0.00049 | 29   | 17   | 0.0114  |
| 8   | 10   | 0.00098 | 29   | 8    | 0.002551 | 8   | 6    | 0.00049 | 29   | 17   | 0.0114  | 9   | 9    | 0.00092 | 30   | 58   | 434.903 |
| 9   | 5    | 0.00039 | 30   | 8    | 0.00289 | 9   | 9    | 0.00092 | 30   | 58   | 434.903 | 10  | 4    | 0.00047 | 31   | 19   | 0.00344 |
| 10  | 16   | 0.0037  | 31   | 53   | 168.747 | 10  | 4    | 0.00047 | 31   | 19   | 0.00344 | 11  | 22   | 0.01392 | 32   | 12   | 0.00514 |
| 11  | 20   | 0.0094  | 32   | 58   | 510.344 | 11  | 22   | 0.01392 | 32   | 12   | 0.00514 | 12  | 9    | 0.00122 | 33   | 10   | 0.00396 |
| 12  | 11   | 0.0018  | 33   | 31   | 1.00335 | 12  | 9    | 0.00122 | 33   | 10   | 0.00396 | 13  | 10   | 0.00145 | 34   | 27   | 0.356276 |
| 13  | 6    | 0.00070 | 34   | 56   | 411.957 | 13  | 10   | 0.00145 | 34   | 27   | 0.356276 | 14  | 1    | 0.00069 | 35   | 43   | 26.97922 |
| 14  | 1    | 0.00060 | 35   | 50   | 124.164 | 14  | 3    | 0.00069 | 35   | 43   | 26.97922 | 15  | 15   | 0.00634 | 36   | 44   | 36.25960 |
| 15  | 17   | 0.01081 | 36   | 47   | 67.4379 | 15  | 15   | 0.00634 | 36   | 44   | 36.25960 | 16  | 4    | 0.00086 | 37   | 62   | 1.74339 |
| 16  | 4    | 0.00086 | 37   | 62   | 174.8339 | 16  | 17   | 0.00706 | 37   | 62   | 1.74339 | 17  | 26   | 0.09706 | 37   | 32   | 1.74117 |
| 17  | 32   | 0.30375 | 38   | 74   | 18096.70 | 17  | 26   | 0.09706 | 37   | 32   | 1.74117 | 18  | 10   | 0.00149 | 41   | 55   | 1143.04 |
| 18  | 17   | 0.0682  | 39   | 56   | 588.9686 | 18  | 11   | 0.00149 | 41   | 55   | 1143.04 | 19  | 28   | 0.15888 | 40   | 13   | 3.16681 |
| 19  | 33   | 0.6984  | 40   | 58   | 951.2177 | 19  | 28   | 0.15888 | 40   | 13   | 3.16681 | 20  | 10   | 0.00149 | 41   | 55   | 1143.04 |
| 20  | 10   | 0.0144  | 41   | 32   | 1.890367 | 20  | 10   | 0.00149 | 41   | 55   | 1143.04 | 21  | 28   | 0.43241 | 42   | 26   | 0.35234 |
| 21  | 28   | 0.1325  | 42   | 55   | 561.6479 | 21  | 28   | 0.43241 | 42   | 26   | 0.35234 | 22  | 14   | 0.0403  | 43   | 46   | 76.0470 |
| 22  | 14   | 0.0403  | 43   | 46   | 76.0470 | 22  | 14   | 0.0403  | 43   | 46   | 76.0470 | 23  | 21   | 0.0376  | 44   | 44   | 48.61702 |
| 23  | 21   | 0.0376  | 44   | 44   | 48.61702 | 23  | 21   | 0.0376  | 44   | 44   | 48.61702 | 24  | 20   | 0.02313 | 45   | 40   | 18.9293 |
| 24  | 20   | 0.02313 | 45   | 40   | 18.9293 |

Table 13: Time taken (in seconds) by Local Search using Maximal Subset (LS MaxS) and Local Search using Minimal Subset (LS MinS) in CA-SSR where \( S_1 \) is randomly chosen and \( \forall n, S_1 \in [1, 2n] \).
### Table 14: Time taken (in seconds) by Basic Bucket Algorithm (Basic BA), Maximum Frequency Driven Bucket Algorithm (Max FD) and Minimum Frequency Driven Bucket Algorithm (Min FD) in CA-SSR where $S_1$ is randomly chosen and $\forall n$, $S_1 \in [1, 2n]$.

| $n$ | $S$ | Time taken (in sec) by Basic BA in CA-SSR | $n$ | $S$ | Time taken (in sec) by Max FD in CA-SSR | $n$ | $S$ | Time taken (in sec) by Min FD in CA-SSR |
|-----|-----|--------------------------------------|-----|-----|--------------------------------------|-----|-----|--------------------------------------|
| 3   | 1   | 0.000551                             | 3   | 1   | 0.00094                             |
| 4   | 1   | 0.00064                             | 4   | 1   | 0.00056                             |
| 5   | 1   | 0.000569                             | 5   | 2   | 0.00072                             |
| 6   | 3   | 0.001332                             | 6   | 4   | 0.00156                             |
| 7   | 6   | 0.0003052                            | 7   | 7   | 0.00073                             |
| 8   | 10  | 0.000837                             | 8   | 1   | 0.00073                             |
| 9   | 12  | 0.05103                              | 9   | 3   | 0.00016                             |
| 10  | 16  | 0.00139                              | 10  | 7   | 0.0384                              |
| 11  | 11  | 0.002192                             | 11  | 20  | 0.12669                             |
| 12  | 11  | 0.00859                              | 12  | 12  | 0.00110                             |
| 13  | 9   | 0.00265                              | 13  | 24  | 0.0003778                           |
| 14  | 16  | 0.00405                              | 14  | 10  | 0.00053                             |
| 15  | 13  | 0.00004                              | 15  | 7   | 0.000849                            |
| 16  | 24  | 0.00119                              | 16  | 12  | 0.00018                             |
| 17  | 24  | 0.00037                              | 17  | 9   | 0.02475                             |
| 18  | 19  | 0.000016                             | 18  | 18  | 0.00717                             |
| 19  | 18  | 4.14428                              | 19  | 9   | 0.00659                             |
| 20  | 15  | 0.00266                              | 20  | 18  | 0.005103                            |
| 21  | 14  | 0.00089                              | 21  | 19  | 0.00026                             |
| 22  | 13  | 11.79708                             | 22  | 24  | 0.00236                             |
| 23  | 16  | 0.00015                              | 23  | 22  | 0.00099                             |
| 24  | 15  | 0.000849                             | 24  | 7   | 0.00170                             |
| 25  | 14  | 0.00016                              | 25  | 22  | 0.00015                             |
| 26  | 13  | 0.00004                              | 26  | 25  | 0.00099                             |
| 27  | 12  | 0.09077                              | 27  | 26  | 0.00170                             |
| 28  | 11  | 1.51263                              | 28  | 27  | 0.00170                             |
| 29  | 10  | 2.3588                               | 29  | 28  | 0.00170                             |
| 30  | 9   | 8.08624                              | 30  | 29  | 0.00170                             |
| 31  | 8   | 241.571                              | 31  | 30  | 0.00170                             |
| 32  | 7   | 1261.39                              | 32  | 31  | 0.00170                             |
| 33  | 6   | 1.86848                              | 33  | 32  | 0.00170                             |
| 34  | 5   | 42.9512                               | 34  | 33  | 0.00170                             |
| 35  | 4   | 21.1548                              | 35  | 34  | 0.00170                             |
| 36  | 3   | 328.4056                             | 36  | 35  | 0.00170                             |
| n  | S  | Time taken (in sec) by Basic BA in CA-LSR |
|----|----|------------------------------------------|
| 4  | 3  | 0.00079                                  |
| 5  | 6  | 0.00102                                  |
| 6  | 4  | 0.00044                                  |
| 7  | 12 | 0.00882                                  |
| 8  | 17 | 0.02263                                  |
| 9  | 18 | 0.11157                                  |
| 10 | 25 | 0.32170                                  |
| 11 | 31 | 1.59752                                  |
| 12 | 35 | 9.34307                                  |
| 13 | 40 | 39.5506                                  |
| 14 | 48 | 437.383                                  |
| 15 | 50 | 1846.33                                  |

Table 15: Time taken (in seconds) by Basic Bucket Algorithm (Basic BA), Maximum Frequency Driven Bucket Algorithm (Max FD) and Minimum Frequency Driven Bucket Algorithm (Min FD) in CA-LSR where \( S_2 \) is randomly chosen and \( \forall n, \ S_2 \in [\text{midSum}(n) - n, \text{midSum}(n)] \).

| n  | S  | Time taken (in sec) by Max FD in CA-LSR |
|----|----|-----------------------------------------|
| 3  | 3  | 0.00143                                  |
| 4  | 5  | 0.00153                                  |
| 5  | 7  | 0.00252                                  |
| 6  | 10 | 0.00522                                  |
| 7  | 14 | 0.01324                                  |
| 8  | 18 | 0.02944                                  |
| 9  | 22 | 0.07514                                  |
| 10 | 27 | 0.15790                                  |
| 11 | 33 | 0.41121                                  |
| 12 | 39 | 1.38157                                  |
| 13 | 45 | 4.16112                                  |
| 14 | 52 | 12.2192                                  |
| 15 | 60 | 39.9648                                  |
| 16 | 68 | 132.079                                  |
| 17 | 76 | 434.065                                  |
| 18 | 85 | 1426.70                                  |
| 19 | 95 | 4850.73                                  |
| 20 | 105| 17189.86                                 |

| n  | S  | Time taken (in sec) by Min FD in Exp-2 |
|----|----|----------------------------------------|
| 3  | 2  | 0.00081                                  |
| 4  | 4  | 0.00129                                  |
| 5  | 3  | 0.00117                                  |
| 6  | 6  | 0.00329                                  |
| 7  | 7  | 0.00451                                  |
| 8  | 10 | 0.01452                                  |
| 9  | 13 | 0.03446                                  |
| 10 | 21 | 0.20658                                  |
| 11 | 31 | 1.87448                                  |
| 12 | 37 | 12.5400                                  |
| 13 | 33 | 7.19542                                  |
| 14 | 46 | 166.192                                  |
| 15 | 67 | 793.294                                  |

Table 16: Time taken (in seconds) by Local Search using Maximal Subset (LS MaxS) and Local Search using Minimal Subset (LS MinS) in CA-LSR where \( S_2 \) is randomly chosen and \( \forall n, \ S_2 \in [\text{midSum}(n) - n, \text{midSum}(n)] \).
| n  | Subsets | Max FD | Min FD | Basic BA | LS MaxS | LS MinS | LDG    | SDG    |
|----|---------|--------|--------|----------|---------|---------|--------|--------|
| 12 | 24      | 67     | 0.195  | 1.822    | 1.618   | 0.019   | 0.016  | 0.00103| 0.00596|
| 12 | 27      | 84     | 0.405  | 2.958    | 2.394   | 0.02    | 0.024  | 0.00116| 0.012512|
| 14 | 28      | 134    | 0.808  | 3.95     | 14.76   | 0.066   | 0.065  | 0.00289| 0.013285|
| 14 | 38      | 274    | 3.9    | 54.175   | 161.639 | 0.215   | 0.256  | 0.00315| 0.00313|
| 15 | 30      | 186    | 1.381  | 7.641    | 21.208  | 0.11    | 0.133  | 0.00499| 0.014521|
| 15 | 45      | 521    | 14.03  | 353.746  | -       | -       | -      | -      |
| 16 | 32      | 253    | 2.341  | 11.761   | 77.48   | 0.211   | 0.255  | 0.00615| 0.017711|
| 16 | 52      | 965    | 45.83  | 1224.328 | -       | -       | -      | -      |
| 17 | 34      | 343    | 4.414  | 27.817   | 236.119 | 0.412   | 0.501  | 0.00821| 0.024524|
| 17 | 59      | 1764   | 153.31 | -        | -       | 10.144  | 6.298  | 0.06963| 0.024976|
| 18 | 36      | 461    | 7.913  | 52.816   | 649.679 | 0.791   | 0.96   | 0.02192| 0.034023|
| 18 | 67      | 3301   | 541.046| -        | -       | 39.129  | 45.385 | 0.00464| 0.0473109|
| 20 | 40      | 806    | 21.923 | 146.823  | -       | 2.81    | 3.438  | 0.04656| 0.055301|
| 20 | 85      | 11812  | 6981.574| -       | -       | 572.839 | 664.875| 0.1357| 0.208991|
| 21 | 42      | 1055   | 38.779 | 268.505  | -       | 5.177   | 6.298  | 0.0664| 0.072871|
| 21 | 94      | 21985  | 25000.63| -       | -       | 2084.648| 2421.476| 0.02268| 4.134605|
| 22 | 34      | 1369   | 4.492  | 84.423   | -       | 9.411   | 11.516 | 0.09218| 0.095904|
| 22 | 104     | 41552  | -      | -        | -       | -       | -      | -      | 0.52493| 8.53939 |
| 25 | 50      | 2896   | 295.741| -        | -       | 52.604  | 64.216 | 0.57455| 0.211722|
| 25 | 137     | 283837 | -      | -        | -       | -       | -      | -      | 2.35755| 73.2227 |
| 27 | 54      | 4649   | 831.93 | -        | -       | 572.839 | 664.875| 0.1357| 0.208991|
| 27 | 162     | 1038222| -      | -        | -       | -       | -      | -      | 10.77463| 345.7571|
| 30 | 60      | 9141   | -      | -        | -       | 733.963 | 897.121| 1.92185| 1.92185|
| 30 | 202     | 7206286| -      | -        | -       | -       | -      | -      | 98.64595| 98.64595|

Table 17: The values of CA-FSV. We fix few pairs of \((n, S_1)\) and \((n, S_2)\) for Max FD, Min FD, Basic BA, LS MaxS, LS MinS, LDG and SDG algorithms where \(S_1 = 2 * n\) and \(S_2 = \text{midSum}(n) - n\). This table shows time taken (in seconds) by all alternate techniques for calculating for these pairs.
Table 18: Comparison of SDG and LDG with naive backtracking algorithm. This table presents the time taken (in sec) while enumerating all $2^n$ subsets of $X_n$ for every value of sum $S$ in range $[0, \frac{n(n+1)}{2}]$. This is the time taken by these algorithms to enumerate each and every subset in CA-SLN.

| n  | SDG   | LDG   | Naive |
|----|-------|-------|-------|
| 22 | 4194304 | 0.44724 | 0.39577 | 364.9816 |
| 23 | 8388608 | 0.77863 | 0.81253 | 689.0156 |
| 24 | 16777216 | 1.64562 | 1.60156 | - |
| 25 | 33554432 | 3.01883 | 3.13995 | - |
| 26 | 67108864 | 6.22996 | 6.21826 | - |
| 27 | 134217728 | 11.55410 | 12.66373 | - |
| 28 | 268435456 | 23.83728 | 25.29129 | - |
| 29 | 536870912 | 46.01338 | 49.41213 | - |
| 30 | 1073741824 | 97.38387 | 98.06444 | - |
| 31 | 2147483648 | 184.78691 | 202.59311 | - |
| 32 | 4294967296 | 375.63728 | 407.96308 | - |
| 33 | 8589934592 | 755.37561 | 844.82139 | - |
| 34 | 17179869184 | 2130.2298 | 2363.4442 | - |

Fig. 4: Plot of SDG, LDG and Naive algorithm while enumerating all $2^n$ subsets of $X_n$ for every value of sum $S$ in range $[0, \frac{n(n+1)}{2}]$. This graph plots time taken by these algorithms to enumerate each and every subset in CA-SLN.
7 Conclusion

Subset Sum Problem, also referred as SSP, is a well-known important problem in computing, cryptography and complexity theory. We extended the traditional SSP and suggested various alternate enumeration techniques. Instead of finding one subset with target sum, we find all possible solution of SSP. Therefore, for \( X = \{5, 4, 9, 11\} \) and \( S = 9 \), the solution to our version of SSP is both \( \{5, 4\} \) and \( \{9\} \). We confined our problem domain by considering first \( n \) natural numbers as set \( X_n \). In other words, we enumerate all \( (2^n - 1) \) power set of a set.

We have analyzed the distribution of \( \mathcal{P}(X_n) \) over sum, length and count of individual elements. We introduced four types of distributions: Sum Distribution, Length Distribution, Length-Sum Distribution and Element Distribution. We extended the concept by explaining their formulae and algorithms, along with illustrations, which showed a definite pattern and relations among these subsets. These distributions are prepossessing procedures for various alternate enumeration techniques for solving SSP.

We developed Backtracking Algorithm (Naive) algorithm. It is an improved and systematic brute force approach for generating various subsets with \( \text{Sum} = S \). Instead of searching exhaustively elements are selected systematically. We iterate through all \( 2^n \) solutions in this an orderly fashion. The inputs for this algorithm are the set of first \( n \) natural numbers \( X_n \) and \( \text{Sum} = S \). Time and space complexities for this algorithm are \( O(n \times 2^n) \) and \( O(n) \) respectively.

We have proposed Subset Generator using Sum Distribution (SDG). This algorithm is a recursive generator based on the concept of Sum Distribution and uses subsets of \( X_{(n-1)} \) to produce results for \( X_n \). This algorithm uses the formula defined in Equation 1. This algorithm is executed using dynamic programming. Subsets of \( X_n \) with \( \text{Sum} = S \) are generated by subsets of \( X_{n-1} \) with \( \text{Sum} = S \) and \( \text{Sum} = (S - n) \). Time and space complexities for this algorithm are \( O(2^n \times n^2) \) and \( O(2^n \times n^2) \) respectively.

We have proposed Subset Generator using Length-Sum Distribution (LDG). This algorithm is a recursive generator based on the concept of Length-Sum Distribution and uses subsets of \( X_{(n-1)} \) to produce results for \( X_n \). This algorithm uses the formula defined in Equation 2. This algorithm is executed using dynamic programming. Subsets of \( X_n \) with \( (\text{Sum} = S, \text{Length} = l) \) are generated by subsets of \( X_{n-1} \) with \( (\text{Sum} = S, \text{Length} = l) \) and \( (\text{Sum} = S - n, \text{Length} = l - 1) \). Time and space complexities for this algorithm are \( O(2^n \times n^2) \) and \( O(2^n \times n^2) \) respectively.

We have also proposed Basic Bucket Algorithm (Basic BA). The basic idea behind this enumeration technique is to use the various distribution values. We consider \( SD[n][S] \) number of empty buckets, storage data structures, and iterate through all elements in descending order. It uses the value of Element Distribution for generating all the desired subsets. During each iteration an element is assigned to one of the buckets. This method is about adding the correct element to the corresponding subset. This is a greedy algorithm. This method uses the concept of lookup table explained in Section 7 and ensures uniqueness among and within the subsets. Time and space complexities for this algorithm are \( O(2^n \cdot n^{-3}) \) and \( O(2^n) \) respectively.

Next, we have extended the concept of Basic Bucket Algorithm (Basic BA) to propose two new bucket algorithms: Maximum Frequency Driven Bucket Algorithm (Max FD) and Minimum Frequency Driven Bucket Algorithm (Min FD). Information used by these recursive algorithms are same as the basic bucket algorithm. For Max FD, instead of choosing elements in descending order, we select maximum element with maximum frequency to generate all \( SD[n][S] \) number of subsets of \( X_n \) with \( \text{Sum} = S \). For Min FD we select maximum element with minimum frequency to generate all \( SD[n][S] \) number of subsets of \( X_n \) with \( \text{Sum} = S \). These methods use the concept of lookup table explained in Section 7 and ensure uniqueness among and within the subsets. Time and space complexities for this algorithm are \( O(2^n \cdot n^{-3}) \) and \( O(2^n) \) respectively.

We have proposed two more algorithms Local Search using Maximal Subset (LS MaxS) and Local Search using Minimal Subset (LS MinS). Maximal and Minimal Subsets are a new idea for categorizing subsets of a given class. First, we divide the power set of \( X_n \), \( \mathcal{P}(X_n) \), on the basis of their sum and then further partition these subsets according to their length. LS MaxS is a heuristic algorithm. It finds all the desired subsets by choosing the maximal subset as the seed. Maximal subset has largest possible element at every position for a given sum(S) and length(l). Therefore, we begin from left most element, decrement the first permissible element followed by increment of next permissible element. LS MinS is also a heuristic algorithm also finds all desired subsets by choosing the minimal subset as the seed. Minimal subset has the smallest possible element at every position for a given sum(S) and length(l). Therefore, we begin from left most element, increment the first permissible element followed by decremental of next permissible element. Every
increment or decrement consists of one unit. Time and space complexities for this algorithm are $O(2^{\sqrt{n}})$ and $O(\frac{2^n}{\sqrt{n}})$ respectively.

**Conjecture** There are algorithms that can enumerate all solutions of Subset Sum Problem for set $X_n$ and sum $S$ where $0 \leq S \leq \frac{n(n+1)}{2}$ with $O(SD[n][S])$ complexity.

An optimal algorithm should enumerate exactly $SD[n][S]$ subsets which are part of the solution.

This work can be extended in following ways:

1. By amortizing and combining different set of sums as one input set. Instead of running one sum at a time, we can group the sum values for running various alternate enumeration techniques. This will save the execution time by avoiding recalculations of subsets for smaller ranges.
2. Additionally, we can reduce the execution time of alternate enumeration techniques. These techniques are implementation and machine dependent. These timings are also data structure dependent. As part of future work, we would like to explore more data structures and more powerful machines to reduce the running times furthermore.
3. We have seen that the Local Search algorithm using Maximal or Minimal Subset comparatively explores less number of extra subsets and have better execution time than bucket algorithms. We can enhance this algorithm by using element distribution to limit the heuristic search, by finding different starting points and applying better distance formula for traversing through the solution space.

**Acknowledgement**

We thank Kannan Srinathan and Geeta Hooda for their discussion on this work.

**Appendix**

**Lookup Technique**

Mapping of each subset with a unique integer is the basic concept used to define a lookup table for power sets of $X_n$, where $X_n = \{1, 2 \ldots n\}$. Lookup table ensures uniqueness among the subsets and within elements for a subset. This table helps us to maintain the uniqueness at runtime of any algorithm. This technique is implemented with the help of bit vectors. Bit vector is a compact data structure which hashes each subset $A = \{A_1, A_2 \ldots A_l\}$ to the corresponding integer, denoted by $num$, $S_{num} = \sum_{i=1}^{l} 2^{A_i} - 1$. We consider a hash of size $2^n$. This hash will maintain a one-to-one mapping between all the subsets of $X_n$ and is denoted by $P(X_n)$.

**Upper Bound on Sum Distribution**

In this section, we use definitions and formulas presented in \[3\] By using the maximum limit on the number of subsets with a particular sum, we find an upper bound of our problem.

$SD[n]$, defined in Section 4.1 represents the count of all the subsets of $X_n$ divided over sum $S$ where $S \in [1, b]$ and $b = \frac{n(n+1)}{2}$ (Table 4). The maximum value of $SD[n]$ is found at $midSum(n) = \lfloor \frac{n(n+1)}{4} \rfloor$. Table 10 represents the value of $SD[n][midSum(n)]$ for first 15 natural numbers.

| $n$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-----|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| $sd[n][midSum(n)]$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 22 | 397 | 722 |

Table 19: Values of $SD[n][midSum(n)]$ for first 15 natural numbers
For each $n$, value of $SD[n][\text{midSum}(n)]$ presented in table 19 is the coefficient of $x^{\frac{n(n+1)}{2}}$ in the expansion of $\{(1 + x)(1 + x^2)(1 + x^3) \ldots (1 + x^n)\}$. This coefficient is denoted as $S(n)$ and $S(n) \approx \sqrt{\frac{2}{\pi}} \cdot 2^n \cdot n^{\frac{3}{2}}$ [16]. Therefore, value of maximum number of subsets with sum as $\text{midSum}(n)$ has exponential bound, $O(2^n \cdot n^{\frac{3}{2}})$. This result is vastly used throughout the thesis in order to find complexities of various enumeration techniques.

References

1. Austrin, P., Kaski, P., Koivisto, M., Miäätä, J.: Space–time tradeoffs for subset sum: An improved worst case algorithm. In: International Colloquium on Automata, Languages, and Programming. pp. 45–56. Springer (2013)
2. Austrin, P., Kaski, P., Koivisto, M., Nederlof, J.: Subset sum in the absence of concentration. In: LIPIcs-Leibniz International Proceedings in Informatics. vol. 30. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2015)
3. Beier, R., Vöcking, B.: Random knapsack in expected polynomial time. In: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing. pp. 232–241. ACM (2003)
4. Fomin, F.V., Kratsch, D.: Exact exponential algorithms. texts in theoretical computer science. an eates series (2010)
5. Galil, Z., Margalit, O.: An almost linear-time algorithm for the dense subset-sum problem. SIAM Journal on Computing 20(6), 1157–1189 (1991)
6. Gilmore, P., Gomory, R.E.: Multistage cutting stock problems of two and more dimensions. Operations research 13(1), 94–120 (1965)
7. Gilmore, P., Gomory, R.: The theory and computation of knapsack functions. Operations Research 14(6), 1045–1074 (1966)
8. Hardy, G.H., Wright, E.M.: An introduction to the theory of numbers. Oxford University Press (1979)
9. Horowitz, E., Sahni, S.: Computing partitions with applications to the knapsack problem. Journal of the ACM (JACM) 22(4), 463–468 (1975)
10. Ibarra, O.H., Kim, C.E.: Fast approximation algorithms for the knapsack and sum of subset problems. Journal of the ACM (JACM) 22(4), 463–468 (1975)
11. Kolliaras, K., Xu, C.: A faster pseudopolynomial time algorithm for subset sum. arXiv preprint arXiv:1507.02318 (2015)
12. Lokshtanov, D., Nederlof, J.: Saving space by algebraization. In: Proceedings of the forty-second ACM symposium on Theory of computing. pp. 321–330. ACM (2010)
13. Pisinger, D.: Linear time algorithms for knapsack problems with bounded weights. Journal of Algorithms 33(1), 1–14 (1999)
14. Schnorr, C.P., Euchner, M.: Lattice basis reduction: improved practical algorithms and solving subset sum problem. Mathematical programming 66(1-3), 181–199 (1994)
15. Schnorr, C.P., Euchner, M.: Lattice basis reduction: improved practical algorithms and solving subset sum problem. Mathematical programming 66(1-3), 181–199 (1994)
16. Sullivan, B.D.: On a conjecture of andrica and tomescu. Journal of Integer Sequences 16(2), 3 (2013)
17. T. Cormen, C. Leiserson, R.R., Stein., C.: Introduction to algorithms. (2014), the MIT Press, 3rd edition
18. Valluri, S.R., Karlapalem, K.: Subset queries in relational databases. arXiv preprint cs/0406029 (2004)
19. Wikipedia: Subset sum problem — wikipedia, the free encyclopedia (2016), [https://en.wikipedia.org/wiki?title=Subset_sum_problem&oldid=736749803] [Online; accessed 29-August-2016]
20. Woeginger, G.J.: Exact algorithms for np-hard problems: A survey. In: Combinatorial OptimizationEureka, You Shrink!. pp. 185–207. Springer (2003)