Design and Evaluation of Alternate Enumeration Techniques for Subset Sum Problem

Thesis submitted in partial fulfillment
of the requirements for the degree of

MS (by Research)
in
Computer Science

by

AVNI VERMA
200902040
avni.verma@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
July 2017

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, "Design and Evaluation of Alternate Enumeration Techniques for Subset Sum Problem" by Avni Verma, has been carried out under my supervision and is not submitted elsewhere for a degree.

31/7/17
Date

Adviser: Kamalakar Karlapalem

Dedicated to my family.

Acknowledgement

I would like to express my deepest gratitude to my advisor Prof. Kamalakar Karlapalem for his invaluable guidance, motivation and support right from the beginning till the end. His dedication in research, new ideas and constant zeal to improve has inspired me all the while and made me understood the true meaning of research. He guided me through every obstacle that I encountered, trusted me at each step and made me become self-dependent for work. The completion of thesis would not have been possible without his guidance, motivation and support.

Secondly, I am highly grateful to Prof. Kannan Srinathan for verifying the preprocessing interactive formulae of my work. I will always be indebted to him for his valuable insights to research.

I would also like to thank my seniors and fellow colleagues at CDE lab. They were always there to have fun, help cope with the work load and teaching me to have faith in oneself for everything.

I would also like to thank my friends Rashmi, Mounica, Pranneetha, Prateek, Arpita, Jaya and Vinay who motivated and encouraged me in my difficult times and were always there to support. They have taught me to work meticulously. Thanks for making all the years of my college life the most memorable and wonderful years of my life. The times that we spent together will always be cherished.

Last but not the least I would like to express my gratitude to my parents, Dr. A.M. Verma and Vinita Verma, my grandfather Late Shri G.C. Shrivastava and my entire family for having faith in me and encouraging me to work hard despite all odds. This thesis would not have been possible without their endless love, affection and support. I can never thank my family enough.

Abstract

The subset sum problem, also referred to as SSP, is an NP-Hard computational problem. SSP has its applications in broad domains like cryptography, number theory, operations 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 thesis, 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 prepossessing 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 extra 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.

Contents

List of Figures

Figure Page

List of Tables

Chapter 1

Introduction

1.1 Introduction

Subset Sum Problem is a well-known problem in computing, cryptography and complexity theory. We also refer to the Subset Sum Problem as SSP. 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. Decision version: Given a set X containing positive integers and a target sum S, is there a subset which sum upto S? This is an 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, an exhaustive search for the solution is practical.[4] Decision version of SSP can be solved by using dynamic programming with time complexity $\mathcal{O}(n \times S)$. This is not counted as polynomial time in complexity as it is not polynomial in the size of the problem.[4]

2. Search version: 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 a 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 thesis, we extend the traditional SSP (Search version) and design various alternate enumeration techniques. Instead of finding one subset with target sum, we find all possible solutions of SSP. For example, $X = \{5, 4, 9, 11\}$ 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 $\frac{n(n+1)}{2}$, $S \in [1, \frac{n(n+1)}{2}]$. Before describing the formulation of our problem in detail we explore the research work conducted in field of SSP.

1.2 Related Work

The Subset Sum Problem has garnered a wide algorithmic discussion and study. The problem solution is standardized as solving in $\mathcal{O}(nu)$ pseudo polynomial time using dynamic programming. It is a part of elementary algorithmic courses where n is the size of set and u is the target sum[1]. There has also been a study on introduction of a new faster pseudo polynomial time algorithm to find whether a subset exists for a given sum provided a set S. Time complexity of this algorithm is $\mathcal{O}(\sqrt{n}t)$, n being the size of the set S and $t \leq u$. This solution is derived from a fast Minkowski sum calculation which in turn is exploited from the subset sum structure of small intervals [10].

The literature also presents a few other algorithms. This includes and FPTAS[2] which is and exact algorithm also providing space and time trade offs [3]. Other algorithms are polynomial time algorithms for low density sums and a non-generic pseudo-polynomial time algorithm with different properties [6, 7, 8, 9].

There are various variants of SSP. However, the variations differ mainly in allowing duplicates in the input as well as subset and expressing the problem as optimization one. Overall, the dense instances are expected to be solved efficiently using dynamic programming while sparse instances offer solutions efficiently using backtracking.

The subset sum problem has seen limited incremental progress over the past few years. The fastest algorithm being in $\mathcal{O}(2^{\frac{n}{2}})$ time dating in 1974 work of Horowitz and Sahini[13]. Thus, despite the vast intuitive coherence of the problem, work towards improving the worst case running time remains a work in progress[12].

The roadblocks and benchmarks associated with the subset sum problem bear close resemblance to the knapsack problem and the classical number theory of partition determination. Knapsack problem does encounter the need for partitioning but differs in the random instantiations considered for subset sum. Gilmore and Gomory are responsible for the early-on progress in knapsack[17, 18]. An algorithm for random knapsack instance solution was presented by Beier and Vocking which worked in polynomial time [15]. Other attempts at various algorithms for variations have been provided in [13]. The classical number theory for partition finding was solved by Hardy and Wright [16] in which they used generating functions. However, the computational scheme used in order to generate these partitions were missing.

Comparatively, there is less amount of work done on enumeration techniques for subset sum problem, which we addressed in this thesis. We have developed different algorithms for alternate enumerations techniques for subset sum problem and have compared their performance.

1.3 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 $\mathcal{P}(X_n) = \{\phi, \{1\}, \{2\} \ldots \{1, 2 \ldots n\}\}$. It is also known as power set. The empty set is denoted as ϕ or $\{\}$ or the null set. In this thesis, we use ϕ for the representation.
   $|\mathcal{P}(X_n)| = a = 2^n$

3. $maxSum(n)$ is the sum of all elements of the universal set X_n. This is the maximum possible sum for any element of $\mathcal{P}(X_n)$.
   $maxSum(n) = b = (1 + 2 + 3 \ldots n) = \frac{n(n+1)}{2}$.
   $Sum(A) \leq maxSum(n) = \frac{n(n+1)}{2} \; \forall A \in \mathcal{P}(X_n)$

4. $Sum(A)$ is the sum of all elements of a set A where A belongs to power set of X_n, $A \in \mathcal{P}(X_n)$.
   - For maintaining consistency throughout the thesis, we assume sum of all elements of ϕ as 0, $Sum(\phi) = 0$.
   - The range of $Sum(A)$ is $[0, \frac{n(n+1)}{2}]$.
   - The minimum possible sum for A, where $A \in \mathcal{P}(X_n)$, is denoted as $minSum(n)$.

5. $midSum(n)$ is the mid point of the range of $Sum(A)$ where $A \in \mathcal{P}(X_n)$. Since, the maximum possible sum for power sets of X_n, $\mathcal{P}(X_n)$ is $\frac{n(n+1)}{2}$ and minimum possible sum is 0, $midSum(n) = \frac{minSum(n)+maxSum(n)}{2} = \frac{0+\frac{n(n+1)}{2}}{2}$
   $midSum = d = \frac{(1+2+3 \ldots n)}{2} = \frac{n(n+1)}{4}$
   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. $Len(A)$ is the count of elements of a set A where A belongs to power sets of X_n, $A \in \mathcal{P}(X_n)$.
   - The range of $Len(A)$ is from 1 to n, $Len(A) \in [1, n]$.
   - We consider, count of elements of subset ϕ as 1. $Len(\phi) = 1$.
   - Therefore, the range of $Len(A)$ is from 1 to n. $Len(A) \in [1, n]$.

7. $minSum(n, l)$ is the sum of a subset A where $A \in \mathcal{P}(X_n)$ 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 $A = \{1, 2 \ldots l\}$.
   $minSum(n, l) = (1 + 2 + \ldots + l) = \frac{l(l+1)}{2}$

8. $maxSum(n, l)$ is the sum of a subset A where $A \in \mathcal{P}(X_n)$ 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, $A = \{n, n-1 \ldots n-(l-1)\}$.
    - $maxSum(n, l) = (n + (n-1) + \ldots + n - l + 1) = n \times l - \frac{(l-1)(l-1+1)}{2}$
    - $maxSum(n, l) = \frac{l(2n-l+1)}{2}$

1.4 Thesis Contribution

- We extend the Subset Sum Problem to enumerate all subsets of X_n with sum S.
- We design distribution formulae for $\mathcal{P}(X_n)$.
    - Sum Distribution is the count of all subsets of X_n with sum S.
    - Length-Sum Distribution is the count of all subsets of X_n with sum S and length l.
    - Element Distribution is the frequency of element e in subsets of X_n with sum S.
- We design algorithms for the distribution formulas. These are the preprocessing procedures which are required for proposing various alternate enumeration techniques for solving SSP.
- We develop seven Alternate Enumeration Techniques for the Subset Sum Problem.
- Extensive experimental studies have been conducted to test the performance of the algorithms under a wide variety of scenarios.
- We conduct a Comparative analysis between these algorithms to justify our motivation. Our approaches are found to be very efficient while exploring the solution space. Compared to backtracking algorithm, our approaches explores much less number of subsets to enumerate subsets of X_n with sum S.
- Without computing limitations,the execution time of our approaches is much smaller than the benchmark (backtracking) algorithm.

1.5 Thesis Organization

In this thesis, we have presented the various enumeration techniques for subset sum problem. The thesis is divided into eight chapters organized as follows:

- Chapter 2 presented the distribution of $\mathcal{P}(X_n)$ over sum, length and count of individual elements. The presented formulas and algorithms, along with examples, showed a definite pattern and relations among these subsets. The chapter also includes theorems and properties which validate our propositions.

- Chapter 3 describes the element distribution and elaborates some examples. We also propose an algorithm for computing element distribution for a given n.

- Chapter 4 proposes three algorithms for enumerating SSP. The first algorithm is the basic algorithm, which is based on backtracking. This is the benchmark algorithm. The second algorithm is based on SD and the last algorithm is based on length-sum distribution. We have also discussed advantages and shortcomings of each.

- Chapter 5 proposes a new algorithm called Basic Bucket Algorithm to construct all subsets of $\mathcal{P}(X_n)$ which sum up to S. Chapter 5 also proposes two another new approaches using Frequency Driven Bucket Algorithms with maximum and minimum frequency as selection criteria.

- Chapter 6 introduces the concept of Maximal and Minimal subset. We propose a subset generation algorithm using Local Search and these two concepts.

- Chapter 7 explains the experimental setup that has been used for all the algorithms for enumeration techniques for SSP. The algorithms are run over different n and sum values. We present the running time for all algorithms and compare their performance.

- Chapter 8 concludes the thesis by summarizing the work done so far. It also includes the future work.

Chapter 2

Sum, Length and Length-Sum Distributions

As stated in Chapter 1 the main aim of this thesis is to develop alternate enumeration techniques for the *Subset Sum Problem*. Henceforth, we will refer to as SSP in this thesis. The fundamental definition of SSP is to find subsets of a certain sum. However, in this thesis we enumerate power set of a set which are categorized based on their sum. We consider the set of natural numbers as our problem domain and present the mathematics for background work to address SSP. In the upcoming sections, we have analyzed the distribution of $\mathcal{P}\left(X_n\right)$ 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 2.1, 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. While many of results might look trivial, we prove these for satisfaction. These distributions, theorems and lemmas are used to present various alternate enumeration techniques for solving SSP in Chapter 4, Chapter 5 and Chapter 6. The formulae are the notation developed in Section-1.3. In Table 2.1, b denotes the maximum possible sum for any element of $\mathcal{P}\left(X_n\right)$, $b = \frac{n(n+1)}{2}$.

Distribution	Formula	Meaning	Value/Assumption
SD Sum-Distribution	A 2D matrix with dimensions $n \times b$, where $\lvert X_n \rvert = n$ and $b = \frac{n(n+1)}{2}$.	$SD[n][S]$ represents the count of all the subsets belonging to $\mathcal{P}\left(X_n\right)$ 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, $SD[n][0] = 1$.

CD Length-Distribution	A 2D matrix with dimensions $n \times n$, where $\|X_n\| = n$.	$CD[n][l]$ represents the count of all the subsets belonging to $\mathcal{P}(X_n)$ of length l. Every row, $CD[n]$, is the length distribution for all subsets of X_n.	In this thesis, the count of number of subsets with 0-length is 1, $Len(\phi) = 1$. $CD[n][0] = 1$ $CD[n][l] = \binom{n}{l},\ \forall l \in [1, n]$.
LD Length-Sum-Distribution	A 3D matrix with dimensions $n \times b \times n$, where $\|X_n\| = n$ and $b = \frac{n(n+1)}{2}$.	$LD[n][S][l]$ represents the count of all the subsets belonging to $\mathcal{P}(X_n)$ with sum S and length l. Every column of this matrix, $LD[n][S][l']$, where $\forall l' \in [0, n]$, is the length distribution for all subsets of X_n with sum S.	Extending the previous assumptions we get, $LD[n][S][0] = 1,\ \forall S \in [0, b]$ $LD[n][0][l] = 1,\ \forall l \in [1, n]$
ED Element-Distribution	A 3D matrix of with dimensions $n \times b \times n$, where $\|X_n\| = n$ and $b = \frac{n(n+1)}{2}$.	$ED[n][S][e]$ represents the count element e in all the subsets belonging to $\mathcal{P}(X_n)$ with sum S. Every row, $ED[n][S]$, is the element distribution for all subsets of X_n with sum S.	In this thesis, we assume the count of element-ϕ in all subsets of $\mathcal{P}(X_n)$ as 0. $ED[n][S][0] = 0,\ \forall S \in [0, b]$ A zero-sum is achieved only by subset ϕ. $ED[n][0][e] = 0,\ \forall e \in [0, n]$.

Table 2.1: Formula, definition, meaning, values and assumptions of all distributions which are required for design and evaluation of alternate enumeration techniques for SSP. First column denotes the distribution name, second and third column define the formula, definition and concept behind every distribution and fourth column states all the assumptions.

2.1 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 2.1 establishes the formula for the sum distribution. In Section 2.1.2, we present the correctness and proof for these formulas. Table 2.2 represents sum distribution for power sets of X_2 and X_3 respectively, $\mathcal{P}(X_2)$ and $\mathcal{P}(X_3)$.

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 ϕ. Since, zero-sum ($Sum = 0$) can be achieved only with subset ϕ and $Sum(\phi)$ is assumed to be 0, as defined in Section 1.3, the count of occurrence of ϕ-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 ϕ, the count of occurrence of ϕ-subset in $P(X_n)$ is taken as 1. Therefore, $SD[n][0] = 1$.

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

$$SD[n][S] = \begin{cases} 1 & (S = 0) \text{ or } (n = 1) \\ SD[n-1][S] & 0 < S < n \\ SD[n-1][S] + SD[n-1][S-n] & n \leq S \leq \lfloor \frac{n(n+1)}{4} \rfloor \\ SD[n][maxSum(n) - S] & \lfloor \frac{n(n+1)}{4} \rfloor < S \leq maxSum(n) = \frac{n(n+1)}{2} \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

2.1.1 Examples of Sum Distribution

In Table 2.3, we present tables representing sum distribution for base cases: X_0, X_1 and X_2. The value for $SD[0][0]$, $SD[1][0]$ and $SD[2][0]$ is considered as 1 in order to cover the subset ϕ. In Table 2.4 we present the subset and sum pairs for X_5 and X_6 respectively. Figure 2.1 shows the plot of number of subsets for the sum range of $X_{10} = \{1, 2, \ldots 10\}$. The sum of all the subsets of $\mathcal{P}\ (X_{10})$ varies form 0 to ($\frac{10(10+1)}{2} = 55$). This graph is symmetric around the $midSum(10) = 28$.

2.1.2 Correctness of the Sum Distribution formula

In this section, we present the theorems and lemmas which prove the correctness of the sum distribution formula, $SD[n][S]$ presented in Equation 2.1.

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

Proof. Let us assume $sum_{(n,S)}$ represents all the subsets of X_n which add up to a sum of S, where n is a natural number and $S \in [0, \frac{n(n+1)}{2}]$. Then, the cardinality of $sum_{(n,S)}$, by definition, is the count of all subsets of X_n with sum S. Therefore,

$$|sum_{(n,S)}| = SD[n][S] \tag{2.2}$$

Value of n	Subset	Sum of the Subset
n=2	ϕ	0
	$\{1\}$	1
	$\{2\}$	2
	$\{1, 2\}$	3
n=3	ϕ	0
	$\{1\}$	1
	$\{2\}$	2
	$\{3\}$	3
	$\{1, 2\}$	3
	$\{1, 3\}$	4
	$\{2, 3\}$	5
	$\{1, 2, 3\}$	6

Table 2.2: Sum of every subset of $\mathcal{P}(X_2)$ and $\mathcal{P}(X_3)$. Every subset of $\mathcal{P}(X_2)$ and $\mathcal{P}(X_3)$ are noted in second column while the corresponding sum of every subset is presented in the third column.

$Sum(S)$	Subsets for $n = 0$	No. of Subsets / Sum Distribution / $SD[0][S]$
0	ϕ	1

$Sum(S)$	Subsets for $n = 1$	No. of Subsets / Sum Distribution / $SD[1][S]$
0	ϕ	1
1	$\{1\}$	1

$Sum(S)$	Subsets for $n = 2$	No. of Subsets / Sum Distribution / $SD[2][S]$
0	ϕ	1
1	$\{1\}$	1
2	$\{2\}$	1
3	$\{1, 2\}$	1

Table 2.3: Sum Distribution for base cases: X_0, X_1 and X_2. First column presents the possible sum values, second column presents the corresponding subsets for a particular sum and the third column presents the sum distribution, $SD[n][S]$.

Sum	Subsets for $n=5$	No. of Subsets / Sum Distribution
0	ϕ	1
1	$\{1\}$	1
2	$\{2\}$	1
3	$\{1,2\}$, $\{3\}$	2
4	$\{1,3\}$, $\{4\}$	2
5	$\{2,3\}$, $\{1,4\}$, $\{5\}$	3
6	$\{1,2,3\}$, $\{2,4\}$, $\{1,5\}$	3
7	$\{1,2,4\}$, $\{3,4\}$, $\{2,5\}$	3
8	$\{1,3,4\}$, $\{1,2,5\}$, $\{3,5\}$	3
9	$\{2,3,4\}$, $\{1,3,5\}$, $\{4,5\}$	3
10	$\{1,2,3,4\}$, $\{1,4,5\}$, $\{2,3,5\}$	3
11	$\{2,4,5\}$, $\{1,2,3,5\}$	2
12	$\{3,4,5\}$, $\{1,2,4,5\}$	2
13	$\{1,3,4,5\}$	1
14	$\{2,3,4,5\}$	1
15	$\{1,2,3,4,5\}$	1

Sum	Subsets for $n=6$	No. of Subsets / Sum Distribution
0	ϕ	1
1	$\{1\}$	1
2	$\{2\}$	1
3	$\{1,2\}$, $\{3\}$	2
4	$\{1,3\}$, $\{4\}$	2
5	$\{2,3\}$, $\{1,4\}$, $\{5\}$	3
6	$\{1,2,3\}$, $\{2,4\}$ $\{1,5\}$, $\{6\}$	4
7	$\{1,2,4\}$, $\{3,4\}$ $\{2,5\}$, $\{1,6\}$	4
8	$\{1,3,4\}$, $\{1,2,5\}$ $\{3,5\}$, $\{2,6\}$	4
9	$\{2,3,4\}$, $\{1,3,5\}$ $\{4,5\}$, $\{3,6\}$ $\{1,2,6\}$	5
10	$\{1,2,3,4\}$ $\{1,4,5\}$ $\{2,3,5\}$, $\{4,6\}$ $\{1,3,6\}$	5
11	$\{2,4,5\}$ $\{1,3,4,5\}$ $\{5,6\}$, $\{1,4,6\}$ $\{2,3,6\}$	5
12	$\{3,4,5\}$ $\{1,2,4,5\}$ $\{1,2,3,6\}$ $\{2,4,6\}$, $\{1,5,6\}$	5
13	$\{1,3,4,5\}$ $\{1,2,4,6\}$ $\{3,4,6\}$, $\{2,5,6\}$	4
14	$\{2,3,4,5\}$ $\{1,3,4,6\}$ $\{1,2,5,6\}$ $\{3,5,6\}$	4
15	$\{1,2,3,4,5\}$ $\{2,3,4,6\}$ $\{1,3,5,6\}$ $\{4,5,6\}$	4
16	$\{2,3,5,6\}$ $\{1,4,5,6\}$ $\{1,2,3,4,6\}$	3
17	$\{2,4,5,6\}$ $\{1,2,3,5,6\}$	2
18	$\{1,2,4,5,6\}$ $\{3,4,5,6\}$	2
19	$\{1,3,4,5,6\}$	1
20	$\{2,3,4,5,6\}$	1
21	$\{1,2,3,4,5,6\}$	1

Table 2.4: Sum Distribution for X_5 and X_6. First column presents the possible sum values, second column presents the corresponding subsets for a particular sum and the third column presents the sum distribution, $SD[n][S]$.

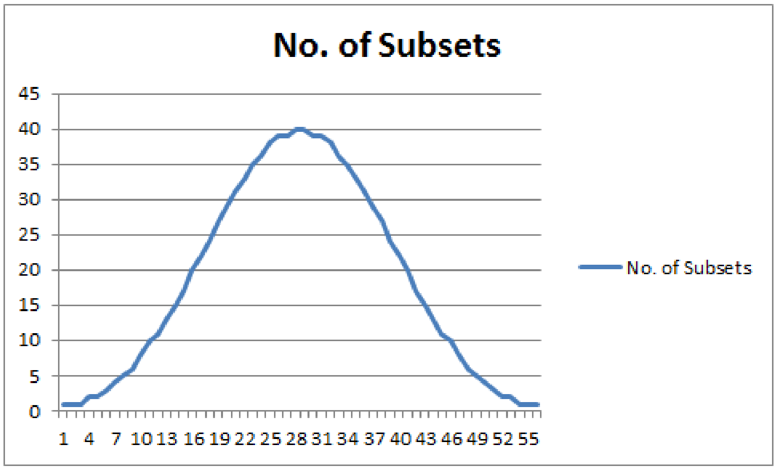


Figure 2.1: Plot of the count of number of subsets of X_n with sum S. On x-axis we have the range of $Sum(A) = [0, 55]$, $A \in \mathcal{P}(X_{10})$ and y-axis denotes the number of subsets of X_{10} with sum S.

Let for some $M \in sum_{(n,S)}$, if $n \in M$, then the remaining elements of the set M are less than or equal to $n - 1$ and the remaining elements sum up to $(S - n)$. Since, all the elements are natural numbers, the sum of these integers belongs to $[0, \frac{n(n+1)}{2}]$ i.e. $(S - n) \geq 0$ or $n \leq S \leq \frac{n(n+1)}{2}$. Therefore,

$$(M - n) \in sum_{(n-1,S-n)} \quad where \quad n \leq S \leq \frac{n(n+1)}{2} \tag{2.3}$$

Similarly, if the element n does not belong to the set M, $n \notin M$, then all the elements of M are less than or equal to $(n - 1)$ and sum up to S. Hence,

$$M \in sum_{(n-1,S)} \quad where \quad n \leq S \leq \frac{n(n+1)}{2} \tag{2.4}$$

From Equation 2.3 and Equation 2.4, we can conclude that all subsets of $sum_{(n,S)}$ are union of two sets: a set of subsets which include element n and a set of subsets which do not include element n. These subsets can be presented as,

$$sum_{(n,S)} = sum_{(n-1,S)} \cup sum_{(n-1,S-n)} \quad where \quad n \leq S \leq \frac{n(n+1)}{2} \tag{2.5}$$

Taking cardinality on both sides and with the use of equation 2.2,

$$|sum_{(n,S)}| = |sum_{(n-1,S)}| + |sum_{(n-1,S-n)}| \quad where \quad n \leq S \leq \frac{n(n+1)}{2} \tag{2.6}$$

$$SD[n][S] = SD[n-1][S] + SD[n-1][S-n] \quad where \quad n \leq S \leq \frac{n(n+1)}{2} \tag{2.7}$$

In order to complete this proof following properties of $sum_{(n,S)}$ should be proved.

1. *Uniqueness:* There should be no duplicate subsets in $sum_{(n,S)}$, $sum_{(n-1,S)} \cap sum_{(n-1,S-n)} = \phi$.

   Proof. $sum_{(n-1,S)}$ is the set of all the subsets of $\mathcal{P}\left(X_{(n-1)}\right)$ with sum S and $sum_{(n-1,S-n)}$ is the set of all the subsets of $\mathcal{P}\left(X_{(n-1)}\right)$ with sum $(S-n)$. We use the method of contradiction to prove set of subsets in $sum_{(n-1,S)}$ and $sum_{(n-1,S-n)}$ are independent. Let us assume subset p belongs to both $sum_{(n-1,S)}$ and $sum_{(n-1,S-n)}$. Since, $p \in sum_{(n-1,S)}$, therefore by definition, the subset p has elements ranging from 1 to $(n-1)$ and all the elements add upto the sum S.

$$S = \sum_{i=1}^{len(p)} p_i \tag{2.8}$$

   Similarly, as per assumption, $p \in sum_{(n-1,S-n)}$. Therefore by definition, the subset p has elements ranging from 1 to $(n-1)$ and all the elements add upto the sum $(S-n)$.

$$(S-n) = \sum_{i=1}^{len(p)} p_i \tag{2.9}$$

   From Equation 2.8 and Equation 2.9 there is a contradiction as $\sum_{i=1}^{len(p)} p_i$ is both S and $(S-n)$. Since, n is a natural number, Equation 2.8 and Equation 2.9 contradict our assumption that a subset p can belong to both sets $sum_{(n-1,S)}$ and $sum_{(n-1,S-n)}$. Therefore, by contradiction, there is no subsets p which belongs to both sets. Hence, $sum_{(n-1,S)}$ and $sum_{(n-1,S-n)}$ are independent. □

2. *Completeness:* $sum_{(n,S)}$ should contain all the subsets of $\mathcal{P}(X_n)$ with sum S.

   Proof. The power set of X_n, $\mathcal{P}(X_n)$, with sum S can be divided in two parts: subsets with sum S which contain element n and subsets with sum S which do not contain element n. By definition, $sum_{(n-1,S)}$ is the set of all the subsets of $\mathcal{P}\left(X_{(n-1)}\right)$ with sum S and $sum_{(n-1,S-n)}$ is the set of all the subsets of $\mathcal{P}\left(X_{(n-1)}\right)$ with sum $(S-n)$.

   In Equation 2.5, the union of sets $sum_{(n-1,S)}$ and $sum_{(n-1,S-n)}$ generates all subsets of $\mathcal{P}(X_n)$ with sum S. Therefore, $sum_{(n,S)}$ should contain all the subsets of $\mathcal{P}(X_n)$ with sum S. □

This completes the statement: $SD[n][S] = SD[n-1][S] + SD[n-1][S-n]$ if $n \leq S \leq \frac{n(n+1)}{2}$.

Since, the graph between the number of subsets in $sum_{(n,S)}$ and the Sum-range $[0, \frac{n(n+1)}{2}]$ is symmetric around $\lfloor \frac{n(n+1)}{4} \rfloor$ we can generate the remaining half values by following the symmetric property. Therefore, we consider S to be less than or equal to $\lfloor \frac{n(n+1)}{4} \rfloor$, i.e. $0 \leq S \leq \lfloor \frac{n(n+1)}{4} \rfloor$.

$$SD[n][S] = SD[n-1][S] + SD[n-1][S-n] \qquad where \qquad n \leq S \leq \lfloor \frac{n(n+1)}{4} \rfloor \tag{2.10}$$

Hence, Equation 2.10 proof the fourth condition of Equation 2.1. □

Lemma 2. *$SD[n][S] = SD[n-1][S]$ if $0 < S < n$.*

Proof. If $S < n$ then, $(S - n) < 0$. Since, $(S - n)$ represents sum of few natural numbers, it cannot be negative and there can be no subsets which add up to this negative sum. Therefore,

$$SD[n-1][S-n] = 0 \tag{2.11}$$

$$SD[n][S] = SD[n-1][S] + 0 \tag{2.12}$$

$$SD[n][S] = SD[n-1][S] \quad where \quad S < n \tag{2.13}$$

Hence, Equation 2.10 proof the third condition of Equation 2.1. □

Theorem 3. *$SD[n][S] = SD[n][maxSum(n) - S]$ if $\lfloor \frac{n(n+1)}{4} \rfloor \leq S \leq maxSum(n) = \frac{n(n+1)}{2}$.*

Proof. Let us assume, $sum_{(n,S)}$ represents all the subsets of X_n of which sum upto S, where $S \in [0, maxSum(n)]$ where $maxSum(n) = \frac{n(n+1)}{2}$. Then, the cardinality of $sum_{(n,S)}$, by definition, is the count of all subsets of X_n of sum S. Therefore,

$$|sum_{(n,S)}| = SD[n][S] \tag{2.14}$$

As specified in Theorem 3, the graph between the number of subsets in $sum_{(n,S)}$ and the sum-range $[0, maxSum(n)]$ is symmetric around $\lfloor \frac{n(n+1)}{4} \rfloor$. We can generate the remaining half of values by following the symmetric property.

$$Value[\lfloor \frac{n(n+1)}{4} \rfloor - x] = Value[\lfloor \frac{n(n+1)}{4} \rfloor + x] \tag{2.15}$$

Using the concepts of set theory, for any set of $M \in P(X_n)$ and universal set U:

$$M + (U - M) = U \tag{2.16}$$

$$M^c = (U - M) \tag{2.17}$$

In Equation 2.17 M^c is the complement set of M. $\forall A \in sum_{(n,S)}$, set A^c has the following properties.

1. n is the largest possible element in subsets of M^c.

2. Sum of all subsets of M^c is $(maxSum(n) - S)$, where $maxSum(n) = \frac{n(n+1)}{2}$, $Sum(A^c) = maxSum(n) - S$

A^c belongs to $sum_{(n,maxSum(n)-S)}$. Therefore, from Equation 2.14,

$$|sum_{(n,maxSum(n)-S)}| = SD[n][maxSum(n) - S] \tag{2.18}$$

From the concepts of set theory for every A there is a complement subset A^c.

$$A \equiv A^c \tag{2.19}$$

$$|sum_{(n,S)}| = |sum_(n, maxSum(n) - S)| \tag{2.20}$$

From Equation 2.14 and Equation 2.20, we can conclude that for $\lfloor\frac{n(n+1)}{4}\rfloor < S \leq maxSum(n) = \frac{n(n+1)}{2}$, $SD[n][S]$ is equal to $SD[n][maxSum(n) - S]$.

$$SD[n][S] = SD[n][maxSum(n) - S] \tag{2.21}$$

Hence, Equation 2.10 proof the last condition of Equation 2.1. □

The bases cases in Section 2.1, Theorem 1, Theorem 1 and Lemma 2 completes the proof of $SD[n][S]$ defined in Equation 2.1.

2.1.3 Sum Distribution: Algorithm and Complexity

In this section, we present Algorithm 1, a pseudo-code to compute Sum distribution for a given n. Since formula stated in Equation 2.1 is recursive, we use a dynamic programming technique to generate desired results. In Line 1 we iterate through all integers between 1 to n. Line 2 and Line 3 define the starting and ending sum for each n. Line 5 and Line 6 calculate the base case when sum is equal to 0, $(S = 0)$. Line 8 counts the subsets without the i^{th} element and Line 11 counts the subsets including i^{th} element. Since, $i \in [1, n]$ and $j \in [0, \frac{n(n+1)}{2}]]$, time complexity of the above algorithm results to $\mathcal{O}(loop_1) * \mathcal{O}(loop_2) = \mathcal{O}(n) * \mathcal{O}(n^2) = \mathcal{O}(n^3)$. Space Complexity for the above algorithm is the size of array storing sum distribution up till n, $SD[n][S]$ is $(n * S)$. Since, $S \in [0, \frac{n(n+1)}{2}]]$ the space complexity result to $\mathcal{O}(n) * \mathcal{O}(S) = \mathcal{O}(n) * \mathcal{O}(n^2) = \mathcal{O}(n^3)$.

Algorithm 1 Sum Distribution(n)

for $i \in \{1, \ldots, n\}$ **do**
    $start\_sum = 0$
    $mid\_sum = \lfloor \frac{i(i+1)}{4} \rfloor$
    $end\_sum = \frac{i(i+1)}{2}$ ▷ end_sum is equal to $maxSum(i)$
    for $j \in \{start\_sum, \ldots, end\_sum\}$ **do**
        if $j == 0$ OR $i == 1$ **then**
            $SD[i][j] = 1$
        else if $j \leq mid\_sum$ **then**
            $SD[i][j] = SD[i-1][j]$
            if $j \geq i$ **then**
                $SD[i][j] += SD[i-1][j-i]$
            end if
        else
            $SD[i][j] = SD[i][end\_sum - j]$
        end if
    end for
end for
return $SD[n]$

2.2 Length Distribution

$\forall M \in P(X_n)$, $Len(M)$ is defined as count of all the elements present in a set M. Since, X_n is the set of first n natural numbers including ϕ, $Len(M) \in [0, n]$. All the subsets of a particular length l are different combinations of l where l is the number of elements chosen from all the n elements, $\binom{n}{l}$ where $l \in [0, n]$. This relation is presented in Equation 2.22. Table-2.5 and Table 2.6 represents count of all subsets for X_4 and X_5 divided on the basis of various possible lengths. Since the number of subsets for each length follows binomial distribution the graph between number of subsets of a particular length is a symmetric curve as shown in Figure 2.2. Length Distribution gave us an idea to relate sum and length parameters of a subset which is explored in detailed in Section 2.3.

$$CD[n][l] = \binom{n}{l} = {}^{n}C_{l} = \frac{n!}{l!(n-l)!} \; where \; l \in [0, n] \tag{2.22}$$

Length	Subsets	No. of Subsets	Length Distribution $\binom{n}{l}$
0	ϕ	1	$\binom{4}{0}$
1	{1},{2},{3},{4}	4	$\binom{4}{1}$
2	{1,2},{1,3},{1,4},{2,3},{2,4},{3,4}	6	$\binom{4}{2}$
3	{2,3,4},{1,3,4},{1,2,4},{1,2,3}	4	$\binom{4}{3}$
4	{1,2,3,4}	1	$\binom{4}{4}$

Table 2.5: Length Distribution for X_4. First column presents the possible length values, second and third columns presents subsets of a particular length and their count respectively and the fourth column shows the length distribution, $CD[n][l]$.

Length	Subsets	No. of Subsets	Length Distribution $\binom{n}{l}$
0	ϕ	1	$\binom{5}{0}$
1	{1},{2},{3},{4},{5}	5	$\binom{5}{1}$
2	{1,2},{1,3},{1,4} {2,3},{2,4},{2,5} {3,4},{3,5},{4,5 {1,5}	10	$\binom{5}{2}$
3	{1,2,3},{1,3,4},{1,4,5} {1,2,5}, {2,3,4},{2,4,5} {2,3,5}, {3,4,5},{1,3,5} {1,4,5}	10	$\binom{5}{3}$
4	{2,3,4,5},{1,3,4,5},{1,2,4,5} {1,2,3,5},{1,2,3,4}	5	$\binom{5}{4}$
5	{1,2,3,4,5}	1	$\binom{5}{5}$

Table 2.6: Length Distribution for X_5. First column presents the possible length values, second and third columns present subsets of a particular length and their count respectively and the fourth column contains the length distribution, $CD[n][l]$.

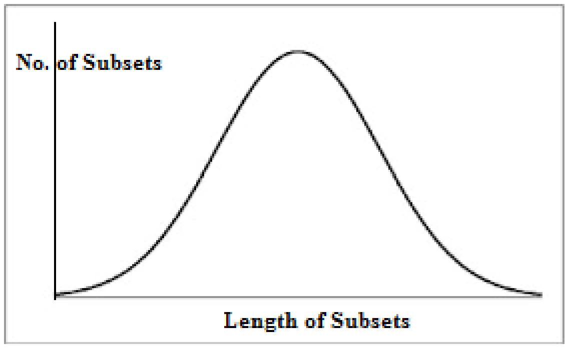


Figure 2.2: Plot of the count of number of subsets of X_n of length l. On x-axis we have the range of $Len(A) = [0, n]$, $A \in \mathcal{P}(X_n)$ and y-axis denotes the number os subsets of X_n of length l. This is a symmetric curve.

2.3 Relation Between Sum Distribution and Length Distribution

In this section, we find a relation between the sum and length distributions of first n natural numbers, $X_n = \{1, 2 \ldots n\}$ as described in Section 2.1 and Section 2.2. Table 2.7 represents ranges of $minSum(n, l)$ and $maxSum(n, l)$ where $l \in [1, n]$, $minSum(n, l)$ and $maxSum(n, l)$ are defined in Section 1.3. Table 2.8 represents length of each subset of $\mathcal{P}(X_4)$, $\mathcal{P}(X_5)$ and $\mathcal{P}(X_6)$ respectively. These tables help us in further analyzing the relation between $Sum(A)$ and $Len(A)$ of a subset, where $A \in \mathcal{P}(X_n)$. On counting the number of the occurrence of each integer for every $sum - length$ pair, we find symmetric patterns which are formulated in more detail as Length-Sum distribution in Section 2.4. These patterns are presented in Table 2.9.

Length	Minimum Sum	Maximum Sum
1	1	5
2	3=(1+2)	9=(5+4)
3	6=(1+2+3)	12=(5+4+3)
4	10=(1+2+3+4)	14=(5+4+3+2)
5	15=(1+2+3+4+5)	15=(5+4+3+2+1)

Table 2.7: Minimum and Maximum Sums for X_5 where length varies from $[1, n]$.

Sum	0	1	2	3	4	5	6	7	8	9	10
$Lengths$	0	1	1	1,2	1,2	2,2	2,3	2,3	3	3	4

Sum	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$Lengths$	0	1	1	1,2	1,2	1,2,2	2,2,3	2,2,3	2,3,3	2,3,3,3	3,3,4	3,4	3,4	4	4	5

Sum	0	1	2	3	4	5	6	7	8	9	10
$Lengths$	0	1	1	1,2	1,2	1,2,2	1,2,2,3	2,2,2,3	2,2,3,3	2,2,3,3,3	2,3,3,3,4
Sum	21	20	19	18	17	16	15	14	13	12	11
$Lengths$	6	5	5	4,5	4,5	4,4,5	3,4,4,5	3,4,4,4	3,3,4,4	3,3,3,4,4	2,3,3,3,4

Table 2.8: $Length$ of each subset against every sum for power set of X_4, X_5 and X_6 respectively.

2.4 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 $S \in [0, maxSum(n)]$, $maxSum(n) = \frac{n(n+1)}{2}$ and $l \in [0, n]$. $LD[n][S][l]$ represents such count. $LD[n][S][l]$ is generated with the help of the symmetric pattern presented in Table 2.9 for $\mathcal{P}(X_4)$, $\mathcal{P}(X_5)$ and $\mathcal{P}(X_6)$ respectively.

$$LD[n][S][l] = \begin{cases} 1 & l = 0 \text{ and } S = 0 \\ LD[n-1][S][l] & 1 \leq l \leq \lfloor \frac{n}{2} \rfloor \text{ and } 0 \leq S < n \\ LD[n-1][S][l] + LD[n-1][S-n][l-1] & 1 \leq l \leq \lfloor \frac{n}{2} \rfloor \text{ and } n \leq S \leq \frac{n(n+1)}{2} \\ LD[n][maxSum(n) - S][n-l] & \lfloor \frac{n}{2} \rfloor < l \leq n \\ 0 & \text{otherwise} \end{cases} \tag{2.23}$$

2.4.1 Examples of Length-Sum Distribution

Table 2.10 presents the bases cases for $Length - Sum$ Distribution and Table 2.11 presents the values of $Length - Sum$ Distribution for $\mathcal{P}(X_5)$ in a simple understandable format. This table maps each subset with length(l) and sum(S).

Sum for $\mathcal{P}(X_4)$	Total Length l	$l=1$	$l=2$	$l=3$	$l=4$
0	1	1	-	-	-
1	1	1	-	-	-
2	1	1	-	-	-
3	2	1	1	-	-
4	2	1	1	-	-
5	2	-	2	-	-
6	2	-	1	1	-
7	2	-	1	1	-
8	1	-	-	1	-
9	1	-	-	1	-
10	1	-	-	-	1

Sum for $\mathcal{P}(X_5)$	Total Length l	$l=1$	$l=2$	$l=3$	$l=4$	$l=5$
0	1	1	-	-	-	-
1	1	1	-	-	-	-
2	1	1	-	-	-	-
3	2	1	1	-	-	-
4	2	1	1	-	-	-
5	3	1	2	-	-	-
6	3	-	2	1	-	-
7	3	-	2	1	-	-
8	3	-	1	2	-	-
9	3	-	1	2	-	-
10	3	-	-	2	1	-
11	2	-	-	1	1	-
12	2	-	-	1	1	-
13	1	-	-	-	1	-
14	1	-	-	-	1	-
15	1	-	-	-	-	1

Sum for $\mathcal{P}(X_6)$	Total Length l	$l=1$	$l=2$	$l=3$	$l=4$	$l=5$	$l=6$
0	1	1	-	-	-	-	-
1	1	1	-	-	-	-	-
2	1	1	-	-	-	-	-
3	2	1	1	-	-	-	-
4	2	1	1	-	-	-	-
5	3	1	2	-	-	-	-
6	4	1	2	1	-	-	-
7	4	-	3	1	-	-	-
8	4	-	2	2	-	-	-
9	5	-	2	3	-	-	-
10	5	-	1	3	1	-	-
11	5	-	1	3	1	-	-
12	5	-	-	3	2	-	-
13	4	-	-	2	2	-	-
14	4	-	-	1	3	-	-
15	4	-	-	1	2	1	-
16	3	-	-	-	2	1	-
17	2	-	-	-	1	1	-
18	2	-	-	-	1	1	-
19	1	-	-	-	-	1	-
20	1	-	-	-	-	1	-
21	1	-	-	-	-	-	1

Table 2.9: $Length - Sum$ Distribution for $\mathcal{P}(X_4)$, $\mathcal{P}(X_5)$ and $\mathcal{P}(X_6)$ respectively

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 $n=1$	Subset	Sum of the Subset	No. of Subsets / Length Distribution
l=0	$\{\phi\}$	0	1
l=1	$\{1\}$	1	1

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

Table 2.10: 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, $LD[n][S][l]$.

Values for n=5														
l=0			l=1			l=2			l=3			l=4		
Sum	Subset	**Size**	Sum	Subset	**Size**	Sum	Subset	**Size**	Sum	Subset	**Size**	Sum	Subset	**Size**
0	ϕ	1	1	{1}	1	3	{1, 2}	1	6	{1, 2, 3}	1	10	{1, 2, 3, 4}	1
			2	{2}	1	4	{1, 3}	1	7	{1, 2, 4}	1	11	{1, 2, 3, 5}	1
			3	{3}	1	5	{1, 4},	2	8	{1, 2, 5}	2	12	{1, 2, 4, 5}	1
							{2, 3}			{1, 3, 4}				
			4	{4}	1	6	{1, 5}	2	9	{2, 3, 4}	2	13	{1, 3, 4, 5}	1
							{2, 4}			{1, 3, 5}				
			5	{5}	1	7	{2, 5}	2	10	{2, 3, 5}	2	14	{2, 3, 4, 5}	1
							{3, 4}			{1, 4, 5}				
						8	{3, 5}	1	11	{2, 4, 5}	1			
						9	{3, 6}	1	12	{3, 4, 5}	1			

l=5		
Sum	Subset	**Size**
15	{1, 2, 3, 4, 5}	1

Table 2.11: Length-Sum Distribution for $\mathcal{P}\left(X_5\right)$

2.4.2 Correctness of the Length-Sum Distribution formula

In this section, we present the theorems and lemma which prove the correctness of $Length - Sum$ distribution formula, $LD[n][S][l]$, presented in Equation 2.23.

Theorem 4. *$LD[n][S][l] = LD[n-1][S][l] + LD[n-1][S-n][l-1]$ if $1 \leq l \leq \lfloor \frac{n}{2} \rfloor$ and $n \leq S \leq \frac{n(n+1)}{2}$.*

Proof. Let us assume, $length_{(n,S,l)}$ represents all the subsets of X_n with length l which sum upto S, $l \in [1, n]$ and $S \in [0, \frac{n(n+1)}{2}]$. Then, the cardinality of $length_{(n,S,l)}$, by definition, is the count of all subsets of X_n with length l and sum S. Therefore,

$$|length_{(n,S,l)}| = LD[n][S][l] \tag{2.24}$$

For $M \in length_{(n,S,l)}$, if the element $n \in M$ then the remaining elements of the set M are less than or equal to $n-1$, their sum is $S-n$ and size of subset $M - \{n\}$ is $(l-1)$. Since, all the elements are natural numbers, the sum of these integers $\in [0, \frac{n(n+1)}{2}]$ i.e. $S - n \geq 0$ or $S \geq n$. Therefore,

$$M - n \in length_{(n-1,S-n,l-1)} \quad where \quad 0 \leq S \leq n \tag{2.25}$$

Similarly, if the element $n \notin M$ then,

$$M \in legth_{(l,n-1,S)} \tag{2.26}$$

From Equation 2.25 and Equation 2.26, we can conclude that all subsets of $length_{(n,S,l)}$ are union of two sets: a set of subsets containing element n and a set of subsets not containing element n, which can be represented as,

$$length_{(n,S,l)} = length_{(n-1,S-n,l-1)} \cup length_{(n-1,S,l)} \quad where \quad 0 \leq S \leq n \tag{2.27}$$

Taking cardinality on both sides and with the use of Equation 2.27,

$$|length_{(n,S,l)}| = |length_{(n-1,S-n,l-1)}| + |length_{(n-1,S,l)}| \quad where \quad 0 \leq S \leq n \tag{2.28}$$

$$LD[n][S][l] = LD[n-1][S][l] + LD[n-1][S-n][l-1] \quad where \quad 0 \leq S \leq n \tag{2.29}$$

In order to complete this proof following properties of $length_{(n,S,l)}$ should be proved.

1. *Uniqueness:* There should be no duplicate subsets in $length_{(n,S,l)}$, $length_{(n-1,S,l)} \cap length_{(n-1,S-n,l-1)} = \phi$.

Proof. $length_{(n-1,S,l)}$ is the set of all the subsets of $\mathcal{P}\left(X_{(n-1)}\right)$ of length l with sum S. $sum_{(n-1,l-1,S-n)}$ is the set of all the subsets of $\mathcal{P}\left(X_{(n-1)}\right)$ of length $(l-1)$ with sum $(S-n)$. We use the method of contradiction to prove set of subsets in $length_{(n-1,S,l)}$ and $length_{(n-1,S-n,l-1)}$ are independent. Let us assume, subset p belongs to both $length_{(n-1,S,l)}$ and $length_{(n-1,S-n,l-1)}$. Since, $p \in length_{(n-1,S,l)}$, therefore by definition, the subset p has l number of elements ranging from 1 to $(n-1)$ and these elements sum upto S.

$$|p| = l \tag{2.30}$$

$$S = \sum_{i=1}^{len(p)} p_i \tag{2.31}$$

Similarly, as per assumption, $p \in length_{(n-1,S-n,l-1)}$. Therefore by definition, the subset p has $l-1$ number of elements ranging from 1 to $(n-1)$ and these elements add upto the sum $(S-n)$.

$$|p| = l - 1 \tag{2.32}$$

$$(S-n) = \sum_{i=1}^{len(p)} p_i \tag{2.33}$$

By using Equations 2.30, 2.31, 2.32 and 2.33,there is a contradiction as $|p|$ is both l and $(l-1)$ and $\sum_{i=1}^{len(p)} p_i$ 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 $length_{(n-1,S,l)}$ and $length_{(n-1,S-n,l-1)}$. Therefore, by contradiction, there is no subsets p which belongs to both sets. Hence, $length_{(n-1,S,l)}$ and $length_{(n-1,S-n,l-1)}$ are independent. □

2. *Completeness:* $length_{(n,S,l)}$ should contain all the subsets of $\mathcal{P}(X_n)$ of length l with sum S.

   Proof. The power set of X_n, $\mathcal{P}(X_n)$ of length l and sum S can be divided in two parts: l length subsets with sum S which contain element n and l length subsets with sum S which do not contain element n. By definition, $length_{(n-1,S,l)}$ is the set of all the subsets of $\mathcal{P}\left(X_{(n-1)}\right)$ of length l with sum S and $length_{(n-1,S-n,l-1)}$ is the set of all the subsets of $\mathcal{P}\left(X_{(n-1)}\right)$ of length $l-1$ with sum $(S-n)$.

   In Equation 2.27, the union of sets $length_{(n-1,S,l)}$ and $length_{(n-1,S-n,l-1)}$ generates all subsets of $\mathcal{P}(X_n)$ of length l with sum S. Therefore, $length_{(n,S,l)}$ should contain all the l length subsets of $\mathcal{P}(X_n)$ with sum S. □

This completes the statement: $LD[n][S][l]] = LD[n-1][S][l] + LD[n-1][S-n][l-1]$ if $0 \leq S \leq n$. Since, the graph between the number of subsets in $length_{(n,S,l)}$ and the length-range $[0, n]$ is symmetric around $\lfloor \frac{n}{2} \rfloor$ we can generate the remaining half values by following the symmetric property. Therefore, we consider l to be less than or equal to $\lfloor \frac{n}{2} \rfloor$, i.e. $1 \leq l \leq \lfloor \frac{n}{2} \rfloor$.

$$LD[n][S][l] = LD[n-1][S][l] + LD[n-1][l-1][S-n] \quad where\ 1 \leq l \leq \lfloor \frac{n}{2} \rfloor\ and\ 0 \leq S \leq n \tag{2.34}$$

Hence proved. □

Lemma 5. *$LD[n][S][l] = LD[n-1][S][l]$ if $1 \leq l \leq \lfloor \frac{n}{2} \rfloor$ and $0 < S < n$.*

Proof. If $S < n$ then, $(S - n) < 0$. Since, $(S - n)$ represents sum of few natural numbers it cannot be negative and there can be no subsets with this negative sum. Therefore,

$$LD[n-1][S-n][l-1] = 0 \tag{2.35}$$

$$LD[n][S][l] = LD[n-1][S][l] + 0 \tag{2.36}$$

$$LD[n][S][l] = LD[n-1][S][l] \quad where 0 < S < n \tag{2.37}$$

Due to symmetric property of the graph between the number of subsets in $length_{(n,S,l)}$ and the length-range $[0, n]$, we consider $1 \leq l \leq \lfloor \frac{n}{2} \rfloor$. Therefore,

$$LD[n][S][l] = LD[n-1][S][l] \quad where \quad 1 \leq l \leq \lfloor \frac{n}{2} \rfloor\ and\ 0 < S < n \tag{2.38}$$

Hence, Equation 2.38 proves the third condition of Equation 2.23.

□

Theorem 6. *$LD[n][S][l] = LD[n][maxSum(n) - S][n - l]$ if $\lfloor \frac{n}{2} \rfloor < l \leq n$, $maxSum(n) = \frac{n(n+1)}{2}$ and $S \leq maxSum(n)$*

Proof. Let us assume, $length_{(n,S,l)}$ represents all the subsets of X_n of length l which sums upto S, where $l \in [1, n]$ and $S \in [0, maxSum(n)]$ and where $maxSum(n) = \frac{n(n+1)}{2}$. Then the cardinality of $length_{(n,S,l)}$, by definition, is the count of all subsets of X_n of length l and sum S. Therefore,

$$|length_{(n,S,l)}| = LD[n][S][l] \tag{2.39}$$

As specified in Theorem 4, the graph between the number of subsets in $length_{(n,S,l)}$ and the length-range $[0, n]$ is symmetric around $\lfloor \frac{n}{2} \rfloor$. We can generate the remaining half of values by following the symmetric property.

$$Value[\lfloor \frac{n}{2} \rfloor - x] = Value[\lfloor \frac{n}{2} \rfloor + x] \tag{2.40}$$

Using the concepts of set theory, for any set of $M \in P(X_n)$ and universal set U:

$$M + (U - M) = U \tag{2.41}$$

$$M^c = (U - M) \tag{2.42}$$

In Equation 2.39 M^c is the complement set of M. $\forall A \in length_{(n,S,l)}$, set A^c has the following properties.

1. Length of all subsets of M^c is $(n - l)$, $|A^c| = (n - l)$
2. n is the largest possible element in subsets of M^c.
3. Sum of all subsets of M^c is $(maxSum(n) - S)$, where $maxSum(n) = \frac{n(n+1)}{2}$, $Sum(A^c) = maxSum(n) - S$

A^c belongs to $length_{(n,maxSum(n)-S,n-l)}$. Therefore, from Equation 2.39,

$$|length_{(n,maxSum(n)-S,n-l)}| = LD[n][maxSum(n) - S][n - l] \tag{2.43}$$

From the concepts of set theory for every A there is a complement subset A^c.

$$A \equiv A^c \tag{2.44}$$

$$|length_{(n,S,l)}| = |length_{(n,maxSum(n)-S,n-l)}| \tag{2.45}$$

From Equation 2.39 and Equation 2.45, we can conclude that for $\lfloor \frac{n}{2} \rfloor < l \leq n$, $maxSum(n) = \frac{n(n+1)}{2}$ and $S \leq maxSum(n)$, $LD[n][S][l]$ is equal to $LD[n][maxSum(n) - S][n - l]$.

$$LD[n][S][l] = LD[n][maxSum(n) - S][n - l] \tag{2.46}$$

Hence, Equation 2.46 proves the fourth condition of Equation 2.23. $\square$

2.4.3 Algorithm and Complexities

In this section, we present Algorithm 2, a pseudo-code to calculate $Length - Sum$ distribution for a given n. In length-sum distribution, formula stated in Equation 2.23 is recursive. Therefore, we use a dynamic technique to generate the desired results. In Line 1 we define the $maxSum$ for a particular n, $maxSum(n) = \frac{n(n+1)}{2}$. It sets the upper limit on our calculations. Line 2 iterates through all integers between 1 to n. Line 3 and Line 4 define the starting and ending sum for each n. From Line 7 to Line 11 we calculate the first half of the values and Line 13 generates the remaining half of the values by using the symmetric property of $Length-Sum$ distribution. Line 8 counts the subsets without i^{th} element and Line 10 counts the subset with i^{th} element. Since, $l \in [1, n]$ and $S \in [0, \frac{n(n+1)}{2}]$ time complexity of the above algorithm results to $\mathcal{O}(loop_1) * \mathcal{O}(loop_2) * \mathcal{O}(loop_2) = \mathcal{O}(n) * \mathcal{O}(n) * \mathcal{O}(n^2) = \mathcal{O}(n^4)$. For the above algorithm, the size of array required to store $Length-Sum$ distribution, $LD[n][S][l]$, is $n * l * S$. $\forall l \in [1, n]$ and $S \in [0, \frac{n(n+1)}{2}]$ the space complexity is $\mathcal{O}(n^2) * \mathcal{O}(S) = \mathcal{O}(n^2) * \mathcal{O}(n^2) = \mathcal{O}(n^4)$.

Algorithm 2 Length Distribution(n, l, S)

1: $LD[0][0][0] = LD[1][0][0] = 1$ ▷ Base Cases
2: $LD[1][1][1] = 1$
3: **for** $i \in \{2, \ldots, n\}$ **do**
4:     $maxSum = \frac{i(i+1)}{2}$
5:     $LD[i][0][0] = 1$
6:     **for** $j \in \{1, \ldots, n\}$ **do**
7:         $startSum = \frac{j(j+1)}{2}$
8:         $endSum = i * j - \frac{i(i-1)}{2}$
9:         **for** $k \in \{startSum, \ldots, endSum\}$ **do**
10:             **if** $j \leq \lfloor \frac{n}{2} \rfloor$ **then**
11:                 $LD[i][j][k] = LD[i-1][j][k]$
12:                 **if** $j \geq 1$ and $k \geq i$ and $i \leq k \leq \frac{i(i+1)}{2}$ **then**
13:                     $LD[i][j][k] += LD[i][j-1][k-i]$
14:                 **end if**
15:             **else**
16:                 $LD[i][j][k] = LD[i][i-j][maxSum-k]$
17:             **end if**
18:         **end for**
19:     **end for**
20: **end for**
21: $Return\ LD[n][l][S]$

2.5 Experimental Result

We have carried out various sets of experiments on an i3-2120 machine with 4GB of RAM to compare and analyze the performance of preprocessing distributions. The values of Sum Distribution, Length

Distribution and Length-Sum Distribution from our experimental results are matching the values calculated by our formulas developed throughout this chapter. These distribution values are only calculated once and stored in suitable data structures for using as input for alternate enumerate techniques. Sum Distribution and Length-Sum Distribution can be easily calculated on this setup till $n = 50$ and $n = 40$ respectively. The time taken for preprocessing is not included in time taken for generating results for alternate enumeration techniques presented in Chapter 7 .

2.6 Summary

All the work we have discussed so far builds the mathematical foundation of this thesis. From the above formule, theorems, lemmas and proofs of Sum Distribution, Length Distribution and Length-Sum Distribution, we show definite patterns and relations among subsets which helps us in developing upcoming alternate techniques for solving Subset Sum Problem. We have also seen the corresponding examples, algorithms and complexities for all the algorithms which generate these distributions. In next chapter we present and establish proofs for Element Distribution and find patterns among subsets based on occurrences of elements.

Chapter 3

Element Distribution

In this chapter, we formulate a new distribution. We explore the idea of counting the number of times an element of X_n occur in a specific class of subsets. These classes of subsets are categorized on the basis of length, sum and few other parameters.

3.1 Introduction of Element Distribution

In Section 2.1, we have explained and explored the concept of Sum Distribution, where we count the number of subsets out of all power set $\mathcal{P}(X_n)$, 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 of X_n, $\forall e \in [1, n]$, $\forall S \in [0, \frac{n(n+1)}{2}]$, element distribution function, $ED[n][S][e]$, is defined as follows:

$$ED[n][S][e] = \begin{cases} 0 & (n = 0) \text{ or } (S = 0) \text{ or } (e = 0) \\ & \text{or } (0 < S < n \text{ and } e == n) \\ ED[n-1][S][e] & 0 \leq S < n \text{ and } 1 \leq e < n \\ ED[n-1][S][e] + ED[n-1][S-n][e] & n \leq S \leq \frac{n(n-1)}{2} \text{ and } 1 \leq e < n \text{ and } n > 2 \\ SD[n-1][S-n] & n \leq S \leq \frac{n(n+1)}{2} \text{ and } e == n \\ SD[n][S] - ED[n][maxSum - S][e] & \frac{n(n-1)}{2} + 1 \leq S \leq \frac{n(n+1)}{2} \text{ and } 1 \leq e < n \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

Element distribution is another prepossessing procedure in which many results may look trivial or straightforward, we prove these for satisfaction. Element distribution's theorems and lemmas are required for presenting various alternate enumeration techniques especially bucket algorithms introduced in Chapter 5.

3.2 Examples of Element Distribution

Table 3.1 represents the count of elements in $\{1, 2\}$ and $\{1, 2, 3\}$ in all subsets of X_2 and X_3 respectively which are divided based on their sums. These are the base cases. Similarly, Table 3.2 and Table 3.3 represents distribution of elements of X_5 and X_6 in $\mathcal{P}(X_5)$ and $\mathcal{P}(X_6)$ respectively, where subsets are categorized on the basis of their Sum. Element distributions of X_0 includes the count of element 0 in subset ϕ with $Sum = 0$. We assume $ED[0][0][0] = 0$. For a given n, the count of element 0 in all the subsets is considered as $NULL$ or 0. We are not including 0 in the set of first n natural numbers. This generate $ED[n][S][0] = 0 \, \forall S \in [0, \frac{n(n+1)}{2}]$. Also, for any value of n, a zero-sum is achieved only by subset ϕ which is an empty set. Therefore, $ED[n][0][e] = 0 \, \forall e \in [0, n]$. We consider values of elements distribution for $\mathcal{P}(X_0)$, $\mathcal{P}(X_1)$ and $\mathcal{P}(X_2)$ as seed values. Following are the values:

1. $ED[0][0][0] = 0$
2. $ED[1][1][1] = ED[2][1][1] = 1$
3. $ED[2][2][2] = 1$
4. $ED[2][3][1] = ED[2][3][2] = 1$
5. otherwise $ED[i][j][k] = 0$

Subsets $\rightarrow$	ϕ	$\{1\}$	$\{2\}$	$\{1, 2\}$
Elements $\downarrow$				
1	0	1	0	1
2	0	0	1	1

Subsets $\rightarrow$	ϕ	$\{1\}$	$\{2\}$	$\{3\}$	$\{1, 2\}$	$\{1, 3\}$	$\{2, 3\}$	$\{1, 2, 3\}$
Elements $\downarrow$								
1	0	1	0	0	1	1	0	1
2	0	0	1	0	1	0	1	1
3	0	0	0	1	0	1	1	1

Table 3.1: Distribution of elements [1,2] in $\mathcal{P}(X_2)$ and elements [1,2,3] in $\mathcal{P}(X_3)$.

Values for n=5																
No. of Subsets for a Sum	1	1	1	2	2	3	3	3	3	3	3	2	2	1	1	1
Sum $\rightarrow$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Integers $\downarrow$																
1	0	1	0	1	1	1	2	1	2	1	2	1	1	1	0	1
2	0	0	1	1	0	1	2	2	1	1	2	2	1	0	1	1
3	0	0	0	1	1	1	1	1	2	2	2	1	1	1	1	1
4	0	0	0	0	1	1	1	2	1	2	2	1	2	1	1	1
5	0	0	0	0	0	1	1	1	2	2	2	2	2	1	1	1

Table 3.2: Distribution of elements $[1, 2, 3, 4, 5]$ in $\mathcal{P}(X_5)$.

Values for n=6																						
No. of Subsets for a Sum	1	1	1	2	2	3	4	4	4	5	5	5	5	4	4	4	3	2	2	1	1	1
Sum →	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Integers ↓																						
1	0	1	0	1	1	1	2	2	2	2	3	2	3	2	2	2	2	1	1	1	0	1
2	0	0	1	1	0	1	2	2	2	2	2	3	2	2	2	2	2	2	1	0	1	1
3	0	0	0	1	1	1	1	1	2	3	3	2	2	2	3	3	2	1	1	1	1	1
4	0	0	0	0	1	1	1	2	1	2	3	2	3	3	2	3	2	1	2	1	1	1
5	0	0	0	0	0	1	1	1	2	2	2	3	3	2	3	3	2	2	2	1	1	1
6	0	0	0	0	0	0	1	1	1	2	2	3	3	3	3	3	3	2	2	1	1	1

Table 3.3: Distribution of elements $[1, 2, 3, 4, 5, 6]$ in $\mathcal{P}\left(X_6\right)$.

3.3 Correctness of the Element Distribution Formula

In this section, we present the theorems and lemma which prove the correctness of Element distribution formula, $ED[n][S][e]$ presented in Equation 3.1. $ED[n][S][e]$ 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 7. *$ED[n][S][n] = 0$ 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 $\mathcal{P}\left(X_n\right)$ 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:

$$c = |element_{(n,S,e)}| \tag{3.2}$$

$$ED[n][S][e] = |element_{(n,S,e)}| \tag{3.3}$$

$$c = ED[n][S][e] \tag{3.4}$$

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.

$$e \in A \tag{3.5}$$

$$Sum(A) \geq e \tag{3.6}$$

$$S \geq e \tag{3.7}$$

Since $(e == n)$ as per the initial conditions, Equation 3.7 will become,

$$S \geq n \tag{3.8}$$

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 3.7

$$ED[n][S][e] = 0 \tag{3.9}$$

$$ED[n][S][e] = 0 \; if \; 0 < S < n \; and \; e == n \tag{3.10}$$

Hence, we have proved the first part of Equation 3.1. □

Theorem 8. *$ED[n][S][e] = ED[n-1][S][e] + ED[n-1][S-n][e]$ 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 upto 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 $\mathcal{P}(X_{n-1})$ which sum upto $(S-n)$ and contain an element e (as presented in Equation 3.11). If $n \notin A$ then, A belongs to the class of subsets of $\mathcal{P}(X_{n-1})$ which sum upto S and contain an element e (as presented in Equation 3.12).

$$A - n \in element_{(n-1,S-n,e)} \tag{3.11}$$

$$A \in element_{(n-1,S,e)} \tag{3.12}$$

From Equation 3.11 and Equation 3.12, we form the set of all subsets which sum up to S and contain element e,

$$element_{(n,S,e)} = element_{(n-1,S,e)} \cup element_{(n-1,S-n,e)} \quad n \leq S \leq \frac{n(n-1)}{2} \quad and \quad 1 \leq e < n \tag{3.13}$$

Taking cardinality on both sides of Equation 3.13,

$$|element_{(n,S,e)}| = |element_{(n-1,S,e)}| + |element_{(n-1,S-n,e)}| \quad n \leq S \leq \frac{n(n-1)}{2} \quad and \quad 1 \leq e < n \tag{3.14}$$

$$ED[n][S][e] = ED[n-1][S][e] + ED[n-1][S-n][e] \quad n \leq S \leq \frac{n(n-1)}{2} \quad and \quad 1 \leq e < n \tag{3.15}$$

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

1. *Uniqueness:* There should be no duplicate subsets in $element_{(n,S,e)}$, $element_{(n-1,S,e)} \cap element_{(n-1,S-n,e)} = \phi$.

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

$$S = \sum_{i=1}^{len(p)} p_i \tag{3.16}$$

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

$$(S-n) = \sum_{i=1}^{len(p)} p_i \tag{3.17}$$

From Equation 3.16 and Equation 3.17, there is a contradiction as $\sum_{i=1}^{len(p)} p_i$ 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 $element_{(n-1,S,e)}$ and $element_{(n-1,S-n,e)}$. Therefore, by contradiction, there is no subsets p which belongs to both sets. Hence, $element_{(n-1,S,e)}$ and $element_{(n-1,S-n,e)}$ are independent. □

2. *Completeness:* $element_{(n,S,e)}$ 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, $element_{(n-1,S,e)}$ is the set of all the subsets of $\mathcal{P}\left(X_{(n-1)}\right)$ with sum S containing element e and $element_{(n-1,S-n,e)}$ is the set of all the subsets of $\mathcal{P}\left(X_{(n-1)}\right)$ with sum $(S-n)$ containing element e.

In Equation 3.13, the union of sets $element_{(n-1,S,e)}$ and $element_{(n-1,S-n,e)}$ generates all subsets of $\mathcal{P}(X_n)$ with sum S containing element e. Therefore, $element_{(n,S,e)}$ 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. $S \geq 0$

$$S \geq 0 \tag{3.18}$$

$$\frac{n(n-1)}{2} - n \geq 0 \tag{3.19}$$

$$\frac{n^2 - n - 2n}{2} \geq 0 \tag{3.20}$$

$$\frac{n^2 - 3n}{2} \geq 0 \tag{3.21}$$

$$\frac{n(n-3)}{2} \geq 0 \tag{3.22}$$

$$n(n-3) \geq 0 \tag{3.23}$$

Therefore, either both n and $n-3$ should be greater than 0 or both should be less than 0. Since, n cannot be negative,

$$n \geq 0 \quad and \quad n \geq 3 \tag{3.24}$$

Therefore,

$$n \geq 3 \tag{3.25}$$

Hence, from Equation 3.15 and Equation 3.25 we have proved the third part of Equation 3.1. □

Lemma 9. *$ED[n][S][e] = ED[n-1][S][e]$ if $0 \leq S < n$ and $1 \leq e < n$.*

Proof. According to Theorem 8,

$$ED[n][S][e] = ED[n-1][S][e] + ED[n-1][S-n][e] \quad n \leq S \leq \frac{n(n-1)}{2} \quad and \quad 1 \leq e < n \tag{3.26}$$

Since,

$$0 \leq S < n \tag{3.27}$$

$$(-n) \leq S - n < 0 \tag{3.28}$$

But a sum cannot be negative. Therefore, count of element e in subsets of $\mathcal{P}\left(X_{n-1}\right)$ which sum up to S is zero, $ED[n-1][S-n][e] = 0$.

$$ED[n][S][e] = ED[n-1][S][e] + 0 \tag{3.29}$$

$$ED[n][S][e] = ED[n-1][S][e] \quad 0 \leq S < n \quad and \quad 1 \leq e < n \tag{3.30}$$

Equation 3.30 proves the second part of Equation 3.1. □

Theorem 10. *$ED[n][S][e] = SD[n-1][S-n]$ if $n \leq S \leq \frac{n(n+1)}{2}$ and $e == n$.*

Proof. Let $element_{(n,S,e)}$ 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 $A \in element_{(n,S,e)}$ and $|element_{(n,S,e)}| = ED[n][S][e] = c$ where $c \geq 0$. Since, element e belongs to set A, $e \in A$,

$$A - e \equiv A - n \in element_{(n-1,S-e,0)} \tag{3.31}$$

Sum S will result in following condition,

$$n \leq S \leq \frac{n(n+1)}{2} \tag{3.32}$$

$$0 \leq S - n \leq \frac{n(n+1)}{2} - n \tag{3.33}$$

Let us assume $S - n$ as S',

$$0 \leq S' \leq \frac{n^2 - n}{2} \tag{3.34}$$

$$0 \leq S' \leq \frac{n(n-1)}{2} \tag{3.35}$$

$$maxSum(n-1) = \frac{n(n-1)}{2} \tag{3.36}$$

From Equation 3.31 and Equation 3.36,

$$\forall A - n \in element_{(n-1,S-n,0)} \equiv element_{(n-1,S-n,e')} \quad where \quad 1 \leq e' < n \tag{3.37}$$

$$\forall A \in element_{(n,S-n+n,n)} \equiv element_{(n-1,S-n,e')} \quad where \quad 1 \leq e' < n \tag{3.38}$$

$$\forall A \in element_{(n,S,n)} \equiv element_{(n-1,S-n,e')} \quad where \quad 1 \leq e' < n \tag{3.39}$$

Taking cardinality on both sides,

$$|element_{(n,S,n)}| = |element_{(n,S,r)}| = |element_{(n-1,S-n,e')}| \tag{3.40}$$

$$ED[n][S][e == n] = ED[n-1][S-n][e'] \tag{3.41}$$

$|element_{(n-1,S-n,e')}|$ 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 2.1 and Equation 3.41,

$$|element_{(n-1,S',e')}| = SD[n-1][S-n] \tag{3.42}$$

$$ED[n-1][S'][e'] = ED[n][S][e == n] = SD[n-1][S-n] \tag{3.43}$$

$$ED[n][S][e] = SD[n-1][S-n] \quad where \quad n \leq S \leq \frac{n(n+1)}{2} \quad and \quad e == n \tag{3.44}$$

Equation 3.44 proves the fourth part of Equation 3.1. □

Theorem 11. *$ED[n][S][e] = SD[n][S] - ED[n][maxSum(n) - S][e]$ 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 1.3. Let us assume $S' = maxSum(n) - S$. Since, the plot between number of subsets and sum follow a symmetric distribution, $SD[n][S]$ will be equal to $SD[n][maxSum(n) - S]$.

$$SD[n][S] = SD[n][S'] = c \tag{3.45}$$

There are c number of subsets which sum up to S and S'. In this case, sum S is greater than the $\frac{maxSum}{2}$ (the mid point) and by using the reflection/symmetric property of the curve we can find all the values of $ED[n][S][e]$.

$$S'' = \frac{maxSum}{2} = \frac{n(n+1)}{4} \tag{3.46}$$

$$S_{low} = \frac{n(n-1)}{2} + 1 \tag{3.47}$$

$$S_{low} - S'' = \frac{n(n-1)}{2} + 1 - \frac{n(n+1)}{4} \tag{3.48}$$

$$S_{low} - S'' = \frac{n(2n-2-n+1)}{2} + 1 \tag{3.49}$$

$$f(n) = S_{low} - S'' = \frac{n^2 - 3n + 4}{4} \tag{3.50}$$

By using the property of second derivative test we show that $f'(n)$ is greater than 0 when $S > \frac{maxSum}{2}$.

$$f'(n) = \frac{d(f(n))}{dn} > 0 \tag{3.51}$$

$$f'(n) = \frac{d(n^2 - 3n + 4/4)}{dn} > 0 \tag{3.52}$$

$$f'(n) = n - \frac{3}{4} > 0 \tag{3.53}$$

$$f'(n) = n > \frac{3}{4} \tag{3.54}$$

Therefore, $\forall n \geq 1$ we can use the symmetric property and calculate half of the values by using the previously calculated values. But for simplicity, values of element distribution for $n = 1$ are covered as the part of base cases and follows symmetric properties for $n \geq 2$.

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

$$A \cup A^c = U \tag{3.55}$$

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 3.55 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 $Count(x, y)$ as the count of element x in any subset or class of subsets y.

$$Count(e, A) + Count(e, A^c) = 1 \tag{3.56}$$

$\forall e \in [1, n], \forall A \in sum_{(n,S)}\ and\ \forall A^c \in sum_{(n,S')}$

$$Count(e, sum_{(n,S)}) + Count(e, sum_{(n,S')}) = |sum_{(n,S)}| * 1 = |sum_{(n,S')}| * 1 \tag{3.57}$$

By using the definition of element distribution and Equation 3.45

$$ED[n][S][e] + ED[n][S'][e] = c \tag{3.58}$$

$$ED[n][S][e] + ED[n][S'][e] = SD[n][S] \tag{3.59}$$

$$ED[n][S][e] = SD[n][S] - ED[n][S'][e] \tag{3.60}$$

Therefore, by putting the value of $S' = (maxSum(n) - S)$

$$ED[n][S][e] = SD[n][S] - ED[n][maxSum(n) - S][e] \tag{3.61}$$

Equation 3.61 proves the last part of Equation 3.1. $\square$

3.4 Algorithm and Complexities

Algorithm 3 presents a pseudo-code to calculate Element distribution for a given n. This algorithm computes the value of $ED[n][S][e]$. Since formula stated in Equation 3.1 is recursive, we used a dynamic technique to generate the desired results. Line 1 iterates through 1 to n which calculates the smaller values that add up to $ED[n][S][e]$. Line 2 defines max_sum and end_sum for a particular integer. max_sum is equal to end_sum. Line 4 and Line 5 iterate over sum S and element e

where $S \in [0, \frac{n(n+1)}{2}]$ and $e \in [1, n]$. In Line 5, we define the base case of $ED[n][S][e]$. Line 6 to Line 16 formalizes various conditions stated in Equation 3.1. For all elements except n^{th} element, Line 9 calculates the Element distribution occurring in subsets with sum S where $S \in [0, n)$, Line 11 counts the occurrences in subsets with sum $n \leq S \leq \frac{n(n-1)}{2}$ and Line 15 calculates the occurrence of these elements when sum is greater than $\frac{n(n-1)}{2}$, $\frac{n(n-1)}{2} \leq S \leq max\_sum$. Line 7 and Line 13 computes the count of n^{th} element for sum is between $[0, n)$ and $[n, max\_sum]$ respectively. Since, iterators $i, k \in [1, n]$ and $j \in [0, n(n+1)/2]$ time complexity of the above algorithm result to $\mathcal{O}(loop_1) * \mathcal{O}(loop_2) * \mathcal{O}(loop_3) = \mathcal{O}(n) * \mathcal{O}(n^2) * \mathcal{O}(n) = \mathcal{O}(n^4)$. The above algorithm requires an array $ED[n][S][e]$ of size $n * n * S$. Since, $S \in [0, n(n+1)/2]$ the space complexity result to $\mathcal{O}(n^2) * \mathcal{O}(S) = \mathcal{O}(n^2) * \mathcal{O}(n^2) = \mathcal{O}(n^4)$.

Algorithm 3 Element Distribution(n, S, e)

1: **for** $i \in \{1 \ldots n\}$ **do**
2: $\quad max\_sum(i * (i+1))/2$
3: $\quad$ **for** $j \in \{0, \ldots, max\_sum\}$ **do**
4: $\quad\quad$ **for** $k \in \{1 \ldots n\}$ **do**
5: $\quad\quad\quad ED[i][j][k] = 0$ ▷ Base Case
6: $\quad\quad\quad$ **if** $0 \leq j < n$ and $k == n$ **then**
7: $\quad\quad\quad\quad ED[i][j][k] = 0$
8: $\quad\quad\quad$ **else if** $0 \leq j < i$ and $1 \leq k < i$ **then**
9: $\quad\quad\quad\quad ED[i][j][k] = ED[i-1][j][k]$
10: $\quad\quad\quad$ **else if** $i \leq j \leq \frac{i(i-1)}{2}$ and $1 \leq k < i$ **then**
11: $\quad\quad\quad\quad ED[i][j][k] = ED[i-1][j][k] + ED[i-1][j-i][k]$
12: $\quad\quad\quad$ **else if** $i \leq j \leq \frac{i(i+1)}{2}$ and $k == i$ **then**
13: $\quad\quad\quad\quad ED[i][j][k] = SD[i-1][j-i]$
14: $\quad\quad\quad$ **else if** $\frac{i(i-1)}{2} + 1 \leq j \leq \frac{i(i+1)}{2}$ and $1 \leq k < i$ **then**
15: $\quad\quad\quad\quad ED[i][j][k] = SD[i][j] - ED[i][maxSum - j][k]$
16: $\quad\quad\quad$ **end if**
17: $\quad\quad$ **end for**
18: $\quad$ **end for**
19: $\quad Free(ED[i-1])$
20: **end for**
21: $Return\ ED[n]$

3.5 Experimental Result

We have carried out various sets of experiments on an i3-2120 machine with 4GB of RAM to compare and analyze the performance of preprocessing distributions. The values of Element Distribution from our experimental results are matching the values calculated by the formula shown in Equation 3.1 and developed throughout this chapter. These distribution values are only calculated once and stored in suitable data structures for using as input for alternate techniques. Element Distribution can be easily calculated on this setup till $n = 40$. The time taken for preprocessing is not included in the time taken for generating results for alternate enumeration techniques presented in Chapter 7.

3.6 Summary

In this chapter, we have explored the core idea for alternate enumeration techniques. Element Distribution is the study of occurrence of each element from set X_n in set M, where $M \in \mathcal{P}(X_n)$. In previous sections, we establish the formula for Element distribution, proofs the correctness by various theorems and lemmas and finally describe the algorithm for calculating the values of Element distribution. Element Distribution is the primary requirement for two major enumeration techniques: Basic Bucket Algorithm and Frequency Driven Bucket Algorithm, explained in Chapter 5.

Chapter 4

Alternate Enumeration Techniques-I

In this and upcoming chapters, 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. The first approach is the backtracking algorithm. It is the basic method of solving SSP. We have treated this algorithm as the benchmark. The second and third approaches are the extension of Sum Distribution (Section 2.1) and Length-Sum Distribution (Section 2.4) respectively. The next three approaches use Sum, Length-Sum and Element Distributions (Section 3.1) for the same.

4.1 Subset Generation using Backtracking

Our aim is to find all the subsets of set X_n with $Sum = S$. According to the exhaustive search algorithm for SSP [4], 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. In Algorithm 4, the primary function, GENERATESUBSETS, takes the set of first n natural numbers set, size of this set n and the desired target sum $targetSum$, as the inputs. A temporary storage array $tuple$, is defined at Line 2 to store and print the resulting subsets. The principal function SUBSETSUM is called from Line 4 with the initial set of parameters. SUBSETSUM is a recursive function which requires the natural number set, storage $tuple$, value of n, current size of tuple $tSize$, current achieved sum $currentSum$, an iterator ite for elements of set and the $targetSum$. The termination condition for this recursive function is achieved when $currentSum$ becomes equal to the $targetSum$. First $tSize$ number of elements of $tuple$ form the resulting subset. Then the algorithm backtracks by excluding the current element and including the next element of set. This condition is called from Line 11 with updated parameters. This exclusion and inclusion allows us to use only one $tuple$ storage to generate all possible subsets with $Sum = targetSum$. If the $currentSum \neq targetSum$, value of $currentSum$ and current element does not exceed our desired sum and we have not exhausted all the elements, then we move along to consider this and the next possible elements by calling SUBSETSUM for all corresponding values. The worst case time complexity for this algorithm is exponential. It is $\mathcal{O}(n \times 2^n)$. The space complexity for this algorithm is the size of $tuple$ array, $\mathcal{O}(n)$.

Algorithm 4 Subset sum Problem using Back tracking

```
function GENERATESUBSETS(set, n, targetSum)
    int tuple[n] = 0
    if ( targetSum ⩾ 1 and targetSum ⩽ n ∗ (n + 1)/2)
        SUBSETSUM(set, tuple, n, 0, 0, 0, tagetSum)
    end if
end function
function SUBSETSUM(set, tuple, n, tSize, currentSum, ite, targetSum)
    if ( targetSum == currentSum)
        PRINTARRAY(tuple, tSize)
        if ( ite + 1 < n and currentSum − set[ite] + set[ite + 1])
            SUBSETSUM(set, tuple, n, tSize − 1, currentSum − set[ite], ite + 1, tagetSum)
        end if
        return
    else
        if ( ite < n and currentSum + set[ite] ⩽ targetSum)
            for i = ite; i <= n; i + +
                tuple[tSize] = set[i]
                if ( currentSum + set[ite] ⩽ targetSum)
                    SUBSETSUM(set, tuple, n, tSize + 1, currentSum + set[ite], ite + 1, tagetSum)
                end if
            end for
        end if
    end if
end function
```

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.

4.2 Subset Generation using Sum Distribution

In the previous chapter (Chapter 2), we establish several concepts, theories and formulas for Sum Distribution. It counts the number of subsets of X_n which sum up to S, where $X_n = \{1, 2, 3 \dots n\}$ and $S \in [0, \frac{n(n+1)}{2}]$, represented by $SD[n][S]$. The recursive equation (Equation 2.1) and dynamic algorithm (Algorithm 1) establishes the theory for sum distribution.

In this section, we design a generator using Sum Distribution. Algorithm 5 is the pseudo-code for generating all the subsets of 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 22 to minimize the space consumption. In Line 2, we iterate through smaller natural 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 21 essentially execute these steps and returns the final result at Line 24.

The value of maximum number of subsets has exponential bound, $\mathcal{O}(2^n * n^{\frac{-3}{2}})$, as described in Appendix C. Therefore, the time complexity for ($loop_3$) at Line 14 is $\mathcal{O}(2^n * n^{\frac{-3}{2}})$. Since, $n \in [1, n]$ and $S \in [0, \frac{n(n+1)}{2}]$, time complexity of the above algorithm results to $\mathcal{O}(loop_1) * \mathcal{O}(loop_2) * \mathcal{O}(loop_3) = \mathcal{O}(n) * \mathcal{O}(n^2) * \mathcal{O}(2^n * n^{\frac{-3}{2}}) = \mathcal{O}(2^n * n^{\frac{3}{2}})$. Space complexity for the above algorithm is the size of array storing smaller subsets, $SDG[n-1][S]$. This complexity is also exponential $n * S * \ No.\ of\ Subsets$. Since $S \in [0, \frac{n(n+1)}{2}]$, the space complexity results to $\mathcal{O}(n) * \mathcal{O}(n^2) * \mathcal{O}(2^n * n^{\frac{-3}{2}})$ i.e. $\mathcal{O}(2^n * n^{\frac{3}{2}})$.

4.3 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 $X_n = \{1, 2, 3 \dots n\}$, $l \in [0, n]$ and $S \in [0, \frac{n(n+1)}{2}]$, represented by $LD[n][S][l]$. The recursive equation (Equation 2.23) and dynamic algorithm (Algorithm 2) establishes the theory for the Length-Sum distribution.

Algorithm 5 GeneratorUsingSumDistribution(n)

```
SDG = {}                                        ▷ Data structure to store the generated Subsets
for i ∈ {1, ..., n} do
    start_sum = 0
    mid_sum = ⌊i(i+1)/4⌋
    end_sum = i(i+1)/2                          ▷ end_sum is equal to maxSum(i)
    universal_set = {1, 2 ... i}                ▷ universal_set is used to calculate the symmetric subsets
    for j ∈ {start_sum, ..., mid_sum} do
        if (j == 0) then
            SDG[i] = {φ}
        end if
        SDG[i][j] = SDG[i − 1][j]
        for subset ∈ SDG[i − 1][j − i] do
            subset.append(i)
            SDG[i][j].append(subset)            ▷ Adding i^th element in every subset of SDG[i − 1][j − i]
        end for
        if j ≠ (i − j) then
            for subset ∈ SDG[i][j] do
                SDG[i][end_sum − j] = universal_set − subset      ▷ Symmetric subsets.
            end for
        end if
    end for
    Free(SDG[i − 1])
end for
return SDG[n]
```

In this section, we present the designed generator. Algorithm 6 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 of length l and sum S where $X_n = \{1, 2, 3 \ldots n\}$, $l \in [0, n]$ and $S \in [0, \frac{n(n+1)}{2}]$ by $LD[n][S][l]$. However, $LDG[i][j][k]$ consists of all subsets of X_i with $length = j$ and $Sum = k$. In LDG notation for length and sum are reversed for easier calculations.

In Algorithm 6, extra values of LDG ($LDG[i-1]$) are freed in Line 28 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 max_sum for X_i, bases cases of $LDG[i][j]$, $start\_sum$ and end_sum. Line 13 to Line 26 iterates through feasible values of sum between $start\_sum$ and 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 i^{th} 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 (end_sum-k) is calculated by subtracting the subset from $universal\_set$. Line 15 to Line 25 essentially execute these steps and returns the final result at Line 30.

The value of maximum number of subsets has exponential bound, $\mathcal{O}(2^n * n^{\frac{-3}{2}})$, as described in Appendix C. Therefore, the time complexity for $(loop_4)$ in Line 16 is $\mathcal{O}(2^n * n^{\frac{-3}{2}})$. Since, $l \in [1, n]$ and $S \in [0, \frac{n(n+1)}{2}]$ time complexity of the above algorithm results to $\mathcal{O}(loop_1) * \mathcal{O}(loop_2) * \mathcal{O}(loop_3) * \mathcal{O}(loop_4)$ i.e. $\mathcal{O}(n) * \mathcal{O}(n) * \mathcal{O}(n^2) * \mathcal{O}(2^n * n^{\frac{-3}{2}}) = \mathcal{O}(n^4 * 2^n * n^{\frac{-3}{2}}) = \mathcal{O}(2^n n^{\frac{5}{2}})$. Space complexity for the above algorithm is the size of array storing smaller subsets, $LDG[n-1]$. This complexity is also exponential $n * l * S * \ (No.\ of\ Subsets)$. Since, $l \in [1, n]$ and $S \in [0, \frac{n(n+1)}{2}]$ the space complexity results to $\mathcal{O}(n) * \mathcal{O}(n) * \mathcal{O}(n^2) * \mathcal{O}(2^n * n^{\frac{-3}{2}}) = \mathcal{O}(n^4) * \mathcal{O}(2^n * n^{\frac{-3}{2}})$ i.e. $\mathcal{O}(2^n * n^{\frac{5}{2}})$.

4.4 Summary

This chapter describes the first three enumeration techniques at length. First we describe the core idea and concept behind backtracking algorithm, also considered as the benchmark algorithm for this thesis. It is an improved and systematic brute force approach. Along with this we explain the algorithm, complexity calculations and drawbacks of backtracking algorithms.

We have also programmed two generators by using the concepts of Sum Distribution and Length-Sum Distribution respectively. These techniques utilize the mathematical theory and achieve desired subsets from power set of X_n. Algorithm and complexities of these generators have also been calculated. All these three algorithms have exponential time and space complexity.

Algorithm 6 GeneratorUsingLengthSumDistribution(n)

$LDG = \{\}$
$LDG[0][0] = LD[1][0] = \{\}$ ▷ Base Cases
$LD[1][1] = \{[1]\}$ ▷ Base Cases
for $i \in \{2, \ldots, n\}$ **do**
  $universal\_set = [1, 2 \ldots n]$ ▷ $universal\_set$ is used to calculate the symmetric subsets
  $max\_sum = \frac{i(i+1)}{2}$
  $LDG[i][0] = \{\}$
  $LDG[i][max\_sum] = \{universal\_set\}$
  for $j \in \{1, \ldots, \frac{i}{2}\}$ **do** ▷ Iterarting till mid point
    $LDG[i][j] = \{\}$
    $start\_sum = \frac{j(j+1)}{2}$
    $end\_sum = i * j - \frac{i(i-1)}{2}$
    for $k \in \{start\_sum, \ldots, end\_sum\}$ **do**
      $LDG[i][j][k] = LDG[i-1][j][k]$
      if $j \geq 1$ and $i \leq k \leq \frac{i(i+1)}{2}$ **then**
        for $subset \in LDG[i-1][j-1][k-i]$ **do**
          $subset.append(i)$ ▷ Adding i^{th} element in every subset of $LDG[i-1][j-1][k-i]$
          $LDG[i][j][k].append(subset)$
        end for
      end if
      if $j \neq (i-j)$ **then**
        for $subset \in LDG[i][j][k]$ **do**
          $LDG[i][i-j][end\_sum - k] = universal\_set - subset$ ▷ Symmetric subsets
        end for
      end if
    end for
  end for
  $Free(LDG[i-1])$
end for
$Return\ LD[n]$

Chapter 5

Alternate Enumeration Techniques-II

5.1 Subset Generation using Basic Bucket Algorithm

In this section, we present a novel 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 B.2. It has been extensively used with this algorithm.

5.1.1 Core Idea for Basic Bucket 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 or multi set. A subset belongs to power sets of X_n $\mathcal{P}(X_n)$.

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 the 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 $undesired$ 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 B.2. 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 num, where $num \in [0, 2^n]$. 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.

5.1.2 Illustrations

In this section, we provide sufficient examples to illustrate the Basic Bucket algorithm of SSP.

5.1.2.1 Illustration-1

We present a detailed explanation of generating all the subsets of X_{10} with sum 15 through the following points.

1. Problem domain for this illustration is $X_{10} = \{1, 2 \ldots 10\}$ and $Sum = 15$.

2. The value of $SD[10][15]$ is 20. This algorithm enumerates 20 subsets of $Sum = 15$.

3. We start with 20 empty buckets as showed in Figure 5.1.

4. We calculate the value of x as the part of initialization step where x denotes the smaller value between the numbers of buckets and n, $n = |X_n|$. Table 5.1 displays the value of x for all the iterations from this round. 0^{th} iteration is the initialization step.

$$x = Min(n, SD[n][S]) = Min(10, 20) = 10 \tag{5.1}$$

Iterations →	0^{th}	1^{st}	2^{nd}	3^{rd}	4^{th}	5^{th}	6^{th}	7^{th}
Value of x	10	10	10	9	8	6	6	5

Table 5.1: Values of x in seven iterations of first round

5. Table 5.2 is the log table for first round of iterations in the Basic Bucket Algorithm.

$Elements \rightarrow$ $Iterations \downarrow$	1	2	3	4	5	6	7	8	9	10
0^{th} $iteration$	9	9	8	8	8	6	5	5	4	3
1^{st} $iteration$	8	8	7	7	7	5	4	4	3	2
2^{nd} $iteration$	7	7	6	6	6	4	3	3	2	1
3^{rd} $iteration$	6	6	5	5	5	3	2	2	1	0
4^{th} $iteration$	5	5	4	4	4	2	1	1	0	0
5^{th} $iteration$	4	4	3	3	3	1	0	0	0	0
6^{th} $iteration$	3	3	2	2	2	1	0	0	0	0
7^{th} $iteration$	3	2	1	1	1	0	0	0	0	0

Table 5.2: Log table for first round of iterations in the basic bucket algorithm. We are generating all 20 subsets of X_{10} with $Sum = 15$.

Table 5.3 presents a detailed description for each iteration of the first round when the Basic Bucket Algorithm is applied for $n = 10$ and $Sum = 15$.

Iterations	**Description**
1	First, we take x largest number of elements from the problem domain X_{10}, and insert them in descending order in first x buckets. This insertion is demonstrated in Figure 5.1. The value of x is 10 for the second time. In this first iteration, number 10 goes to bucket $B1$, number 9 goes to bucket $B2$, number 8 goes to bucket $B3$ and so on till number 1 is stored in bucket $B10$.

2	In second iteration, we again consider x (value of x is 10) largest numbers from the universal set X_{10}. Only those elements are taken into consideration for which the current value of $ED[n][S][e]$ is greater than 0, $ED[10][15][e] > 0 \; \forall \; e \in [1, n]$. Elements considered for the current iteration are $\{10, 9, 8 \ldots 1\}$. Figure 5.2 shows how the elements are filled in corresponding buckets. 1. Number 10 placed in bucket $B6$ making the sum of bucket $B6$ to 15. This inclusion creates the first subset of X_{10} belonging to $sum_{(10,15)}$. $B6$ is the first choice of keeping number 10 as bucket $B1$ already contains one occurrence of number 10 and on adding 10 to any bucket between $B2$ and $B5$ will make the final sum greater than 15. 2. In the same way, number 9 is placed in bucket $B5$, next to number 6 resulting in generation of another subset of X_{10} with sum 15. $B5$ is the first bucket where the final sum will not cross 15 and does not contain any other occurrence of number 9. 3. We assign remaining elements to various buckets. All the placements are properly marked by horizontal arrows in figure 5.2. 4. At the end of second iteration, four subsets are successfully generated: $\{\{10, 4, 1\}, \{7, 8\}, \{6, 9\}, \{5, 10\}\}$. The resulting buckets are: $B1, B4, B5$ and $B6$. They are represented in Figure 5.2 by double lined buckets.
7	Figure 5.3 shows the state of the algorithm at the end of first round finally. 1. First round consist of seven iterations. Fourteen subsets with sum 15 are generated. All these buckets are marked by double lines. 2. There are four buckets: $B13, B16, B19, B20$, which contain unique elements with sum less than 15. These buckets are represented by single lines. 3. Last category of buckets: $B3$ and $B7$, are marked by dotted lines. These are $undesired$ subsets. They have unique elements. They require number 1 to attain the sum 15. But, adding number 1 to these buckets will result in duplication. Hence, these buckets are marked differently.

Table 5.3: Elaborates on initial few iterations of first round of the Basic Bucket Algorithm for $n = 10$ and $Sum = 15$.

Table 5.4 and Table 5.5 are the log table for second and third round of iterations for the Basic Bucket Algorithm calculating subsets of X_{10} with $Sum = 15$. Similarly, Figure 5.4 and Figure 5.5(a) shows

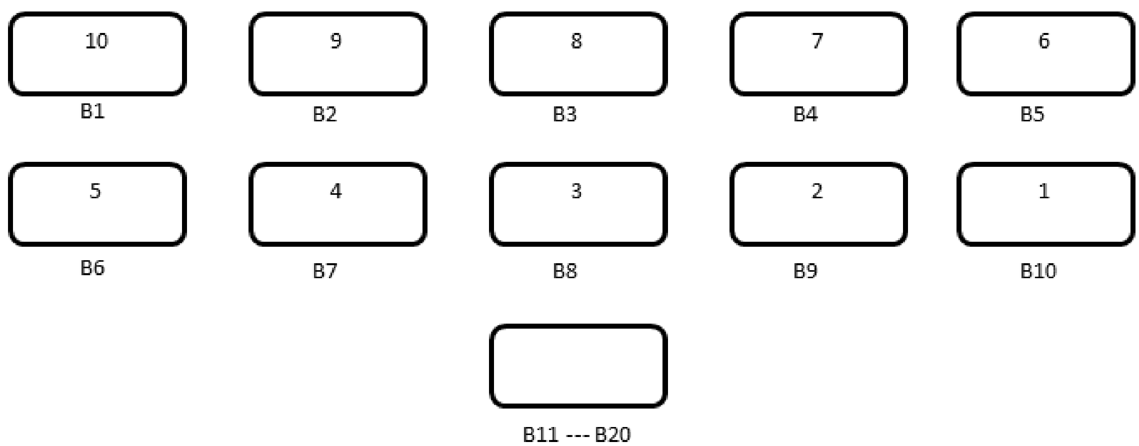


Figure 5.1: Initialization of buckets with first ten natural numbers while enumerating subsets of X_{10} with $Sum = 15$. The last bucket $B11$ to $B20$ denotes ten empty buckets.

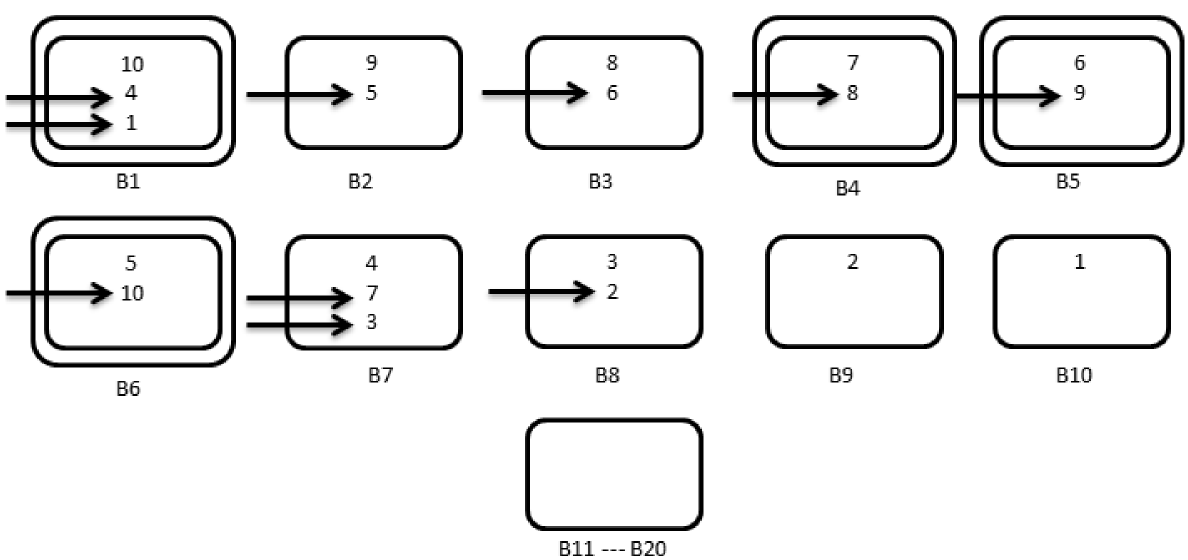


Figure 5.2: Filling buckets in first iteration for the first round of Basic Bucket Algorithm. Generating subsets for first ten natural numbers, X_{10} with $Sum = 15$

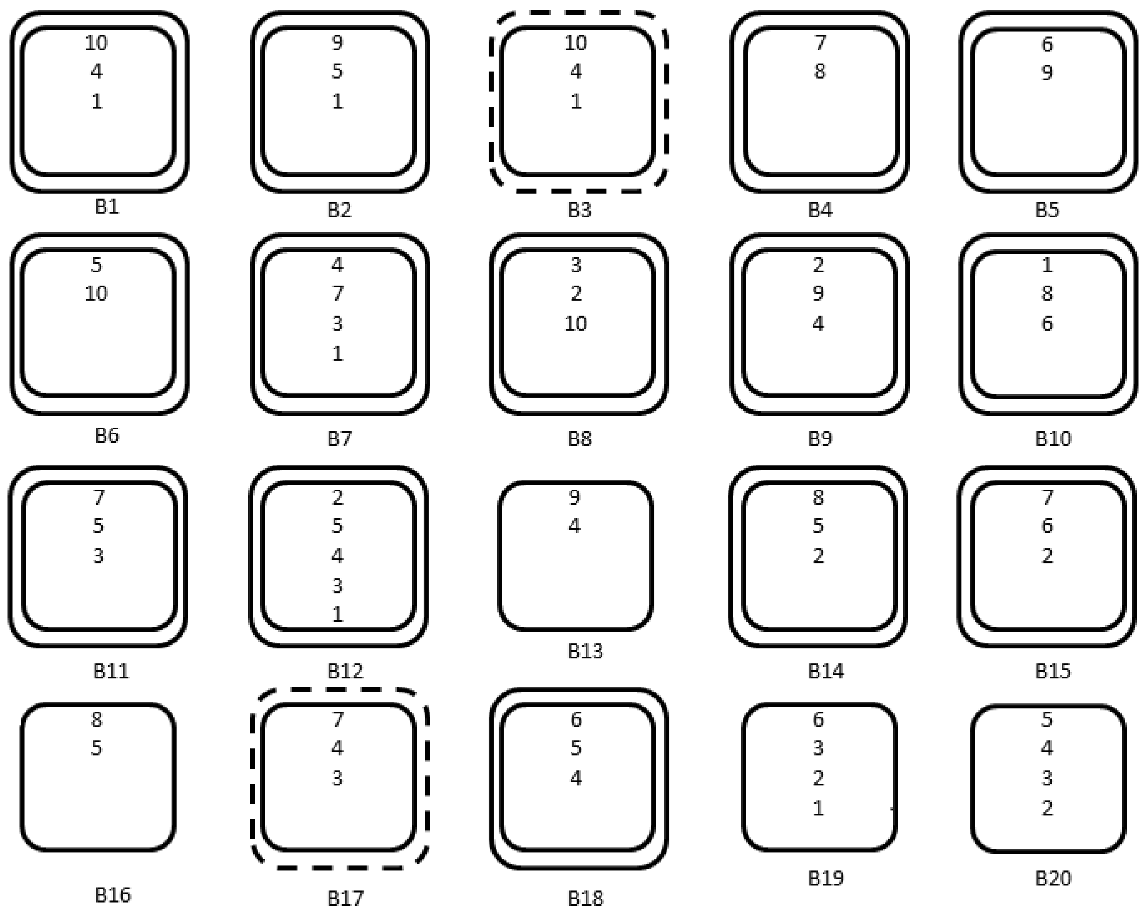


Figure 5.3: Filling buckets in final iteration i.e. iteration 7 for the first round of Basic Bucket Algorithm. Generating subsets for first ten natural numbers, X_{10} with $Sum = 15$

$Elements \rightarrow$ $Iterations \downarrow$	1	2	3	4	5	6	7	8	9	10
0^{th} $iteration$	4	4	4	3	2	2	1	2	1	0
1^{st} $iteration$	3	3	3	2	1	1	0	1	0	0
2^{nd} $iteration$	2	2	2	1	0	0	0	0	0	0
3^{rd} $iteration$	1	2	1	0	0	0	0	0	0	0

Table 5.4: Second round of iterations for Illustration of Basic Bucket Algorithm, an alternate enumeration technique to solve SSP.

filling of buckets in final iteration i.e iteration 3 for second and third rounds of Basic Bucket Algorithm respectively.

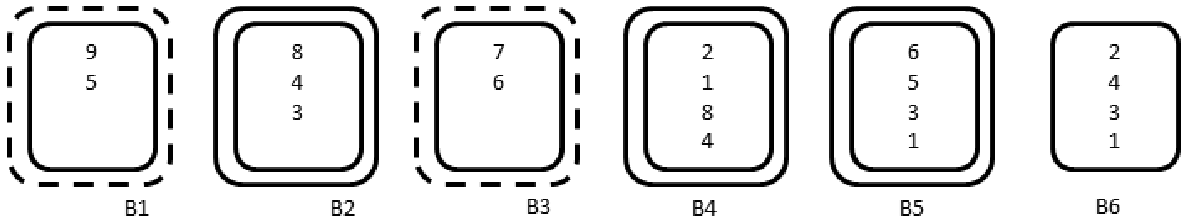


Figure 5.4: Filling buckets in final iteration i.e. iteration 3 for the second round of Basic Bucket Algorithm. Generating subsets for first ten natural numbers, X_{10} with $Sum = 15$

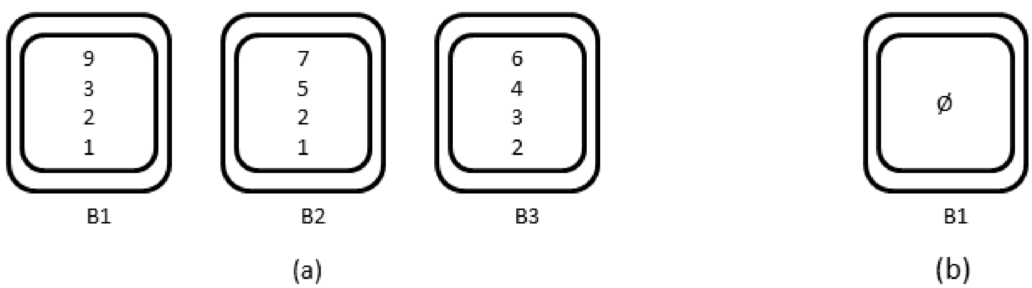


Figure 5.5: (a) Filling ten buckets in final iteration i.e. iteration 3 for the third round of Basic Bucket Algorithm. Generating subsets for first 10 natural numbers, X_{10} with $Sum = 15$. (b) Filling one bucket in a single iteration for the only round of Basic Bucket Algorithm while generating subsets for first 6 natural numbers, X_6 with $Sum = 0$. The resulting bucket contains ϕ.

Figures/Tables	**Descriptions**
Figure 5.5(b) and Table 5.7	These present the complete information about generating subset ϕ for X_6. This is filling one bucket in a single iteration for the only round of Basic Bucket Algorithm while generating subsets for first 6 natural numbers, X_6 with $Sum = 0$. The resulting bucket contains ϕ. Log table for the single iteration is also presented. Count of all first 6 elements is nil when resulting sum is 0.

Figure 5.6	This figure shows the filling of a bucket in a six different iterations under a single round of Basic Bucket Algorithm while generating subsets for first 6 natural numbers, X_6 with $Sum = 21$. A single bucket is properly filled in one round. An element is placed in the bucket, following descending order, in every iteration. The resulting bucket contains the universal set of X_6 which add up to a sum of 21.
Table 5.8	The log table for six iterations in one round for generating the single subset of X_6 of sum 21.

Table 5.6: Illustration of Basic Bucket Algorithm for X_6 with $sum = 0$ and 21.

5.1.2.2 Illustration-2

For sake of completeness and exhaustiveness, we present another illustration for X_6 with $sum = 0$ and 21. This is formation of empty subset and the universal set respectively.

Element	1	2	3	4	5	6
0^{th} iteration	1	1	1	1	1	1
1^{st} iteration	1	1	1	1	1	0
2^{nd} iteration	1	1	1	1	0	0
3^{rd} iteration	1	1	1	0	0	0
4^{th} iteration	1	1	0	0	0	0
5^{th} iteration	1	0	0	0	0	0
6^{th} iteration	0	0	0	0	0	0

Table 5.8: Log table for six iterations in one round for generating one subset of X_6 which sum up to 21 in Basic Bucket Algorithm.

5.1.3 Algorithm and Complexities

In this section we present the pseudo code for the Basic Bucket Algorithm, along with other useful sub-functionalities. Algorithm 7 calculates the element distribution before start of each round of Basic Bucket Algorithm. Algorithm 8 initializes the buckets at the start of the algorithm. It finds the value of x, defined in Equation 5.1, and accordingly fill the buckets with the starting elements. This method is called from Line 3 of the function GENERATINGSUBSETS($n, S, SD[n][S], ED[n][S], prevWrongSubsets$) from the main Algorithm 10. We find an appropriate bucket for every element based on the properties

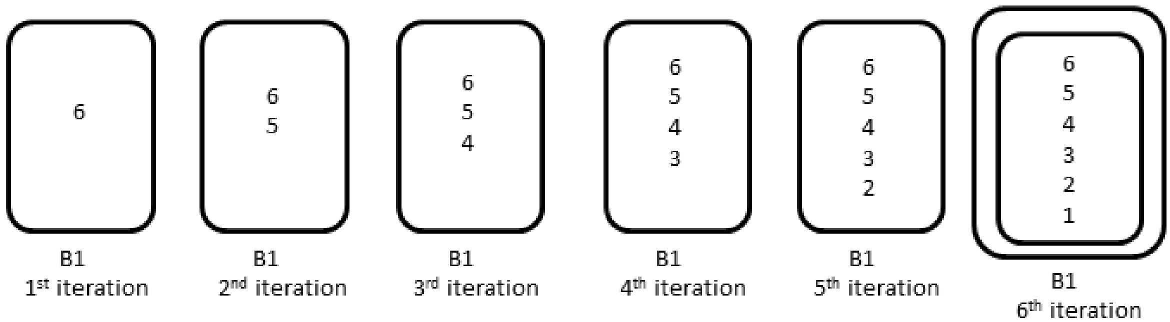


Figure 5.6: Filling buckets in six iterations for the only round of Basic Bucket Algorithm while generating subsets for first six natural numbers, X_6 with $Sum = 21$. A single bucket is completely filled in a single round. The resulting bucket contains the universal set of X_6 which add up to a sum of 21.

$Elements \rightarrow$ $Iterations \downarrow$	1	2	3	4	5	6	7	8	9	10
0^{th} $iteration$	2	3	2	1	1	1	1	0	1	0
1^{st} $iteration$	1	2	1	0	0	0	0	0	0	0
2^{nd} $iteration$	0	1	0	0	0	0	0	0	0	0
3^{rd} $iteration$	0	0	0	0	0	0	0	0	0	0

Table 5.5: Third round of iterations for Basic Bucket Algorithm, an alternate enumeration algorithm for solving SSP.

Element	1	2	3	4	5	6
0^{th} iteration	0	0	0	0	0	0

Table 5.7: Log table for the single iteration in Basic Bucket Algorithm is presented above. Count of all first six elements is nil when resulting sum is 0.

of the Basic Bucket Algorithm. Functionality is defined in Algorithm 9. While Algorithm 11 iterates though all the rounds of the bucket algorithm. All iterations of every round is implemented by the Algorithm 10.

Algorithm 7 GetED($n, S, Table, wrongSubsets$)

function GETED($n, S, Table, wrongSubsets$)
  $newTable = Table$
  for $subset \in wrongSubsets$ **do**
    for $ele \in subset$ **do**
      $newTable[ele]+ = 1$ ▷ Restoring all the ellments of $wrongSubsets$ to the element distribution
    end for
  end for
  $Return\ newTable$
end function

Algorithm 8 InitializeBuckets($all\_buckets$, $Table$, n, S , p)

function INITIALIZEBUCKETS($all\_buckets$, $Table$, n, S , p)
  $q =$ count of non-zero entries of $Table$
  $x = min(p, q)$
  $elements = x$ largest integers of X_n where $Table[ele] \neq 0\ \forall\ ele \in elements$
  Sort $elements$ in descending order
  $bucketIndex = 1$
  for ele in $elements$ **do**
    Add ele in $all\_buckets[bucketIndex]$
    $bucketIndex + +$
  end for
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, $\mathcal{O}(2^n * n^{\frac{-3}{2}})$, as described in Appendix C, time complexity is $\mathcal{O}(max(SD[n][S] \cdot max(SD[n][S]) = \mathcal{O}(2^n * n^{\frac{-3}{2}} \cdot 2^n * n^{\frac{-3}{2}}) = \mathcal{O}(2^{2n} \cdot n^{-3})$. Therefore, given n and S, the time complexity to generate all the subsets is $\mathcal{O}(2^{2n} \cdot n^{-3})$. Space complexity includes size of two storages $Table$ and $all\_buckets$, $\mathcal{O}(n) + \mathcal{O}(2^n) = \mathcal{O}(2^n)$.

5.2 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, $SD[n][S]$, in these two algorithms we use element distribution in decreasing or increasing order. In

Algorithm 9 FindBucket($all\_buckets$, ele, S)

```
function FindBucket(all_buckets, ele, S)
    for bucket in all_buckets do
        if any bucket entry is same as ele then
            Next
        else if on adding ele in bucket, Sum(bucket) > S then
            Next
        else if on adding ele in bucket, bucket becomes duplicate to any other bucket then
            Next
        else
            Return bucket
        end if
    end for
    Return False
end function
```

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.

5.2.1 Core Idea for Frequency Driven 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 5.1.1.

5.2.2 Illustration

In this section, we generate all twenty subsets of X_{10} with $Sum = 15$. 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 5.9, we log all the iterations for generating all twenty subsets of X_{10} with $Sum = 15$. 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 5.10. All subsets are generated in eight iterations.

Algorithm 10 Generating Subsets($n, S, SD[n][S], ED[n][S], prevWrongSubsets$)

Given: n, S, $SD[n][S]$ and $ED[n][S][i]$ where $i \in [1, n]$
$desiredSubsets = [\,]$ ▷ $desiredSubsets$ are all the subsets of X_n with sum S.
$wrongSubsets = [\,]$ ▷ $wrongSubsets$ are the set of $undesired$ and $smaller$subsets.
$Table = GetED(n, S, ED[n][S], prevWrongSubsets)$ ▷ the count of every element in subsets of X_n with sum S called from function $GetED$ described in Algorithm 7
$p = SD[n][S]$: number of subsets of X_n with sum S

function GENERATESUBSETS
  $all\_buckets = p$ empty buckets
  INITIALIZEBUCKETS($all\_buckets$, $Table$, n, S, p) ▷ Initial Step
  $fillBuckets = True$ ▷ Flag to control implementation of the while loop
  while ($fillBuckets$ is set & ($|all\_buckets| > 0$)) **do**
    $filBuckets = False$
    $q =$ count of non-zero entries of $Table$
    $x = min(p, q)$
    $elements = x$ largest integers of X_n where $Table[ele] \neq 0 \; \forall \; ele \in elements$
    Sort $elements$ in descending order
    for $ele \in elements$ **do**
      $b =$ FINDBUCKET($all\_buckets$, ele, S)
      if b is a bucket **then** ▷ When an elemnt can be inserted in a valid bucket.
        Add ele in bucket b
        $fillBuckets = True$ ▷ If no element is alloted to any bucket in a full iteration.
        $Table[ele] - -$
        if Sum of the bucket $b == S$ **then**
          $desiredSubsets + = b$
          print bucket b
          Remove b from $all\_buckets$
          $|all\_buckets| - -$
        end if
      end if
    end for
  end while
  for $bucket \in remaining\_buckets$ **do**
    $wrongSubsets + = bucket$
  end for
  $Return\ wrongSubsets,\ Table$
end function

Algorithm 11 main Function(n, S)

```
function MAINFUNCTION(n, S)
    prevWrongSubsets = [ ]
    prevTable = ED[n][S]
    countSubsets = SD[n][S]
    while countSubsets > 0 do
        prevWrongSubsets, prevTable = GeneratingSubsets(n, S, countSubsets,
                                                prevTable, prevWrongSubsets)
        countSubsets = SD[n][S] = |prevWrongSubsets|
            ▷ Count of Subsets to be generated in next round is same as the size of wrong no. of subsets
    from previous round.
    end while
    Return True
end function
```

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 $max(1,2)$, 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 5.9 and Table 5.10 presents the log entries and subsets corresponding to all iterations of maximum FD bucket algorithm for X_{10} with $sum = 15$.

5.2.3 Algorithm and Complexities

We state the pseudo codes for solving minimum and maximum FD bucket algorithms. Algorithm 12 updates the element distribution after every iteration and is called from Algorithm 14. 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 13, repeatedly calls the $GeneratingSubsetsbyFDBucketAlgo$ function and updates following information:

Elements $\rightarrow$ *Iterations* $\downarrow$	1	2	3	4	5	6	7	8	9	10
0^{th} *iteration*	9	9	8	8	8	6	5	5	4	**3**
1^{st} *iteration*	8	8	7	7	7	6	5	5	**4**	0
2^{nd} *iteration*	6	6	6	6	6	5	5	**5**	0	0
3^{rd} *iteration*	4	4	5	4	5	4	**4**	0	0	0
4^{th} *iteration*	2	**2**	3	3	3	3	0	0	0	0
5^{th} *iteration*	1	0	1	**1**	2	2	0	0	0	0
6^{th} *iteration*	1	0	1	0	1	**1**	0	0	0	0
7^{th} *iteration*	0	0	0	0	0	0	0	0	0	0

Elements $\rightarrow$ *Iterations* $\downarrow$	1	2	3	4	5	6	7	8	9	10
0^{th} *iteration*	9	**9**	8	8	8	6	5	5	4	3
1^{st} *iteration*	5	0	4	4	**5**	4	3	3	2	2
2^{nd} *iteration*	3	0	2	3	0	2	2	**3**	1	1
3^{rd} *iteration*	2	0	1	**2**	0	1	1	0	1	1
4^{th} *iteration*	0	0	0	0	0	1	0	0	**1**	0
5^{th} *iteration*	0	0	0	0	0	0	0	0	0	0

Table 5.9: 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*	$\lvert Subsets \rvert$
1^{st} *iteration*	10	$\{\{10,5\},\{10,4,1\},\{10,3,2\}\}$	3
2^{nd} *iteration*	9	$\{\{9,6\},\{9,5,1\},\{9,4,2\},\{9,3,2,1\}\}$	4
3^{rd} *iteration*	8	$\{\{8,7\},\{8,6,1\},\{8,5,2\},\{8,3,4\},\{8,4,2,1\}\}$	5
4^{th} *iteration*	7	$\{\{7,6,2\},\{7,5,3\},\{7,5,2,1\},\{7,4,3,1\}\}$	4
5^{th} *iteration*	2	$\{\{2,6,4,3\},\{2,5,4,3,1\}\}$	2
6^{th} *iteration*	4	$\{\{4,6,5\}\}$	1
7^{th} *iteration*	6	$\{\{6,5,3,1\}\}$	1
Total Number of Subsets $\rightarrow$			20

Iterations	*Selected Element*	*Subsets*	$\lvert Subsets \rvert$
1^{st} *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
2^{nd} *iteration*	5	$\{\{4,5,6\},\{1,5,9\},\{10,5\},\{7,5,3\},\{1,3,5,6\}\}$	5
3^{rd} *iteration*	8	$\{\{4,3,8\},\{1,6,8\},\{8,7\}\}$	3
4^{th} *iteration*	4	$\{\{4,1,3,7\},\{4,1,10\}\}$	2
5^{th} *iteration*	9	$\{\{9,6\}\}$	1
Total Number of Subsets $\rightarrow$			20

Table 5.10: 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.

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 14. 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. 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, $\mathcal{O}(2^n * n^{\frac{-3}{2}})$, as described in Appendix C, time complexity is $\mathcal{O}(max(SD[n][S] \cdot max(SD[n][S]) = \mathcal{O}(2^n * n^{\frac{-3}{2}} \cdot 2^n * n^{\frac{-3}{2}}) = \mathcal{O}(2^{2n} \cdot n^{-3})$. Therefore, for given n and S,the time complexity to generate all the subsets is $\mathcal{O}(2^{2n} \cdot n^{-3})$. Space complexity includes size of two storages $Table$ and $desiredSubsets$, $\mathcal{O}(n) + \mathcal{O}(2^n) = \mathcal{O}(2^n)$.

Algorithm 12 GetED($n, S, Table, desiredSubsets$)

```
function GETED(n, S, Table, desiredSubsets)
    newTable = Table
    for subset ∈ desiredSubsets do
        for ele ∈ subset do
            newTable[ele]− = 1        ▷ Reducing count of elements according to desiredSubsets.
        end for
    end for
    Return newTable
end function
```

Algorithm 13 main Function(n, S)

```
function MAINFUNCTION(n, S)
    fullTable = ED[n][S]
    countSubsets = SD[n][S]
    elements = [1, 2 ... n]                          ▷ Available elements
    elementsCovered = [ ]                            ▷ Elements covered so far
    while countSubsets > 0 do
        desiredSubsets = GeneratingSubsets(n, S, countSubsets,
                                 elements, elementsCovered, fullTable)
        countSubsets = SD[n][S] − |desiredSubsets|
        Update fullTable, elements, elementsCovered
                 ▷ Reduce frequency of elements according to desiredSubsets.
    end while
    Return True
end function
```

5.3 Summary

Basic Bucket algorithm and Frequency Driven Bucket algorithms with maximum and minimum frequency as selection criteria are three very important alternate enumeration techniques for SSP. First we covered Basic Bucket Algorithm, its core idea, examples, algorithm and complexity analysis in detail. Even though this algorithm is iterative with an exponential complexity, it a new interesting method of solving SSP. Sum and Element distributions provide a methodical way to insert elements in empty subsets covering them to desired subsets with $Sum = S$.

We have also extended the previous algorithm to design Frequency Driven Bucket algorithm. Instead of assigning elements to buckets or subsets in descending order, we assign elements based on their frequency. In every iteration the maximum element with maximum or minimum frequency are chosen. In next chapter, we present two more approaches for enumerating SSP.

Algorithm 14 GeneratingSubsetsbyFDBucketAlgo($n, S, SD[n][S], elements, elementsCovered, ED[n][S]$)

```
Given: n, S, SD[n][S] and ED[n][S][i] where i ∈ [1, n]
desiredSubsets = [ ]                    ▷ desiredSubsets are all the subsets of X_n with sum S.
fullTable = GetED(n, S, ED[n][S], desiredSubsets)    ▷ the count of every element in subsets
    of X_n with sum S called from function GetED described in Algorithm 12
p = SD[n][S] : number of subsets of X_n with sum S

function GenerateSubsets
    if S <= 0 or |elements| == 0 then
        Return desiredSubsets
    end if
    countSubsets = SD[n][S]
    while countSubsets > 0 do
        minVal = min(ED[n][S][e] > 0)
        minKey = max(e ∀e ∈ [1, n] & ED[n][S][e] == minVal)
        elements.remove(minKey)
        elementsCovered.add(minKey)
        desiredSubsets = GeneratingSubsetsbyFDBucketAlgo(n,
                    S − minKey, countSubsets, elements, elementsCovered, fullTable)
        if desiredSubsets[S − minKey] is empty then
            countSubsets − −
            ED[n][S][minKey] − −
            desiredSubsets[S] = [[minKey]]
            print(desiredSubsets[S])
            elementsCovered.remove(minKey)
        else
            for A ∈ desiredSubsets[S − minKey] do
                ED[n][S][minKey] − −
                if (minKey ∉ A) & (minKey + sum(A) == S) then
                    if A.append(minKey) is unique then
                        countSubsets − −
                        print(A)
                        desiredSubsets[S].append(A)
                        In elementsCovered add elements of A
                    end if
                end if
            end for
        end if
    end while
    for e ∈ ED[n][S][e] <= 0 do
        elementsCovered.add(e)
    end for
    for A ∈ desiredSubsets & e ∈ A do
        ED[n][S][e] + +
    end for
    Return desiredSubsets
end function
```

Chapter 6

Alternate Enumeration Techniques-III

6.1 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.

6.1.1 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, $\mathcal{P}(X_n)$, 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 2.4.

6.1.1.1 Definitions for Maximal and Minimal Subsets

For defining maximal subset we have the set of first n natural numbers, X_n, sum(S) which belongs to $[0, maxSum(n)]$ where $maxSum(n) = \frac{n(n+1)}{2}$ and length(l) which belongs to $[0, n]$. Consider, A denotes the set of 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 $A_{i,j}$ 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, $A_{maximal}$, is defined such that $\forall j \in [1, l]$ $A_{maximal,j} > A_{p,j}$ where $p \in \{[1, k] - \{maximal\}\}$. There also exists a minimal subset, $A_{minimal}$, defined such that $\forall j \in [1, l] A_{minimal,j} < A_{p,j}$ where $p \in \{[1, k] - \{minimal\}\}$.

In order to generate the subset $A_{maximal}$ 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. Elements of maximal and minimal subsets are arranged in an ascending order.

Table 6.1 and Table 6.2 displays the maximal and minimal subsets for every sum and length pair of X_4 and X_5 respectively.

Sum	$Length$	$Subsets$	$MaximalSubset$	$MinimalSubset$
0	0	ϕ	ϕ	ϕ
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	2	$\{\{2,3\},\{1,4\}\}$	$\{2,3\}$	$\{1,4\}$
6	2	$\{\{2,4\}\}$	$\{2,4\}$	$\{2,4\}$
	3	$\{\{1,2,3\}\}$	$\{1,2,3\}$	$\{1,2,3\}$
7	2	$\{\{3,4\}\}$	$\{3,4\}$	$\{3,4\}$
	3	$\{\{1,2,4\}\}$	$\{1,2,4\}$	$\{1,2,4\}$
8	3	$\{\{1,3,4\}\}$	$\{1,3,4\}$	$\{1,3,4\}$
9	3	$\{\{2,3,4\}\}$	$\{2,3,4\}$	$\{2,3,4\}$
10	4	$\{\{1,2,3,4\}\}$	$\{1,2,3,4\}$	$\{1,2,3,4\}$

Table 6.1: Maximal and minimal subsets for every sum and length pair of X_4.

6.1.1.2 Algorithms for Maximal and Minimal Subsets

Algorithm 15 and Algorithm 16 generates maximal and minimal subsets of X_n for a given sum S and length l, symbolized by $A_{maximal}$ and $A_{minimal}$ respectively. The base case of the algorithms is achieved when the desired length is one and desired sum is less than or equal to n. While calculating maximal or minimal subset, we iterate through all possible lengths in descending order. MAXIMALSUBSETFUNC in Algorithm 15 and MINIMALSUBSETFUNC in Algorithm 16 returns an element a for the l^{th} position.

For MAXIMALSUBSETFUNC, we first find the minimum possible sum, x, for the rightmost position. x is the smaller number between x_1 and x_2. x_1 is the quotient of sum S and length l which denotes one of the possible sum for l^{th} position defined at line 16. x_2 is the smallest possible entry

Sum	$Length$	$Subsets$	$MaximalSubset$	$MinimalSubset$
0	0	ϕ	ϕ	ϕ
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 6.2: Maximal and minimal subsets for every sum and length pair of X_5.

for l^{th} position. For every possible sum, iterated by i between x and n in ascending order, we find the $maxSum$ for n, S, l, i and $minSum$ for these variables. If sum(S) falls within the range of $minSum$ and $maxSum$, then we return i, element at l^{th} position of resulting maximal subset, $A_{maximal}$. Similarly, for MINIMALSUBSETFUNC in Algorithm 16, the length l, is iterated in descending order. We first find the maximum possible sum, x, for the rightmost position. Next, we define $startPt$ which is the quotient of sum(A) and length (l). It denotes one of the possible sum for l^{th} position. $endPt$ is the largest possible entry for l^{th} position. For every possible sum, iterated by i between $endPt$ and $startPt$ in descending order, we find the $maxSum$ for n, S, l, i and $minSum$ for these variables. If sum(s) falls within the range of $minSum$ and $maxSum$, then Line 25 returns i, element at l^{th} position of resulting minimal subset, $A_{minimal}$. The values of $maxSum$ and $minSum$ are calculated similarly, as calculated by FINDMAXSUM and FINDMINSUM which are defined in Algorithm 15.

Line 29 to Line 35 of FINDMAXSUM, subset $\{i-l+1, i-l+2, \dots i-1, i\}$ of length l adds up to $maxSum$ and in FINDMINSUM line 36 to line42 subset $\{1, 2 \dots (l-1)+i\}$ of length l adds up to $minSum$.

Time complexity of these algorithms is the complexity of the for loop in MAXIMALSUBSET or MINIMALSUBSET function and the complexity of the for loop in MAXIMALSUBSETFUNC or MINIMALSUBSETFUNC function. Therefore, given n, S and l time complexity to generate maximal/minimal subset is $\mathcal{O}(n^2)$. Space complexity is the size of space required to store the subset, $\mathcal{O}(n)$.

6.1.1.3 Experimental Result

We have carried out various sets of experiments on an i3-2120 machine with 4GB of RAM to generate maximal and minimal subsets for X_n with $Sum = S$ where $S \in [1, \frac{n(n+1)}{2}]$ and $l \in [1, n]$. Time taken for generating maximal and minimal subsets for X_n with $Sum = S$ where $S \in [1, \frac{n(n+1)}{2}]$, $l \in [1, n]$ and $n \in [1, 74]$ is less than 5 seconds each. We have calculated these values only till $n = 74$ as the number of subsets of X_n with S where $S \in [1, \frac{n(n+1)}{2}]$ and $n > 74$ exceeds the integer range. The time taken for calculating maximal and minimal subset is not included in time taken for generating results for Local Search algorithm in Chapter 7.

Algorithm 15 Maximal Subset

Given: Natural number n, Sum s and Length l
function MAXIMALSUBSET(n, s, l)
    $maximalSet = [\,]$
    $sum = s$
    $prev = n$
    if $(l == 1)$ and $(sum \leq n)$ **then** ▷ Base Case
        $maximalSet+ = sum$
    end if
    for $i = l; i \geq 2; i - -$ **do**
        $a = maximalSubsetFunc(n, sum, i, prev)$
        $sum- = a$
        $maximalSet+ = a$
        $prev = a - 1$
    end for
    return $maximalSet$
end function

function MAXIMALSUBSETFUNC(n, s, l, $prev$)
    $x_1 = \lfloor s/l \rfloor$
    $x_2 = prev$ ▷ entry at $(l+1)^{th}$ position -1 OR n in case of last element. This is the smallest element that can be alloted as l^{th} element of this subset
    $start\_Pt, end\_Pt = x_1 < x_2?(x_1, x_2) : (x_2, x_1)$
    ▷ Goal is to find smallest and the largest element at l^{th} position which contributes to sum $= s$ and length $= l$.
    for $i \in \{start\_Pt \ldots end\_Pt\}$ **do**
        $maxSum = findMaxSum(l, i)$
        $minSum = findMinSum(l, i)$
        if $minSum \leq s \leq maxSum$ **then**
            return i ▷ i is an element
        end if
    end for
    return $False$
end function

function FINDMAXSUM(l, i)
    ▷ Consider element i at l^{th} position. maxSum will be $\{(i-l+1), \ldots (i-2), (i-1), i\}$
    $maxSum = i + (i-1) + (i-2) + \ldots + (i-l+1)$
    $maxSum = i * l - (1 + 2 + \ldots (l-1))$
    $maxSum = i * l - \frac{l(l-1)}{2}$
    return $maxSum$
end function

function FINDMINSUM(l, i)
    ▷ Consider element i at l^{th} position. minSum will be $\{1, 2 \ldots (l-1), i\}$
    $minSum = 1 + 2 + \ldots + (l-1) + i$
    $minSum = i + (1 + 2 + \ldots (l-1))$
    $minSum = i + \frac{l(l-1)}{2}$
    return $minSum$
end function

Algorithm 16 Minimal Subset

Given: Natural number n, Sum s and Length l
function MINIMALSUBSET(n, s, l)
    $minimalSet = [\,]$
    $sum = s$
    $prev = n$
    if $(len == 1)$ and $(sum \leq n)$ **then** ▷ Base Case
        $minimalSet+ = sum$
    end if
    for $i = l; i \geq 2; i - -$ **do**
        $a = minimalSubsetFunc(n, sum, i, prev)$
        $sum- = a$
        $minimalSet+ = a$
        $prev = a - 1$
    end for
    return $minimalSet$
end function

function MINIMALSUBSETFUNC(n, s, l, $prev$)
    $x_1 = \lfloor s/l \rfloor$
    $x_2 = prev$ ▷ largest possible element at l^{th} position
    ▷ Goal is to find largest element at l^{th} position which contributes to sum $= s$ and length $= l$.
    $start\_Pt, end\_Pt = x_1 < x_2?(x_1, x_2) : (x_2, x_1)$
    ▷ Goal is to find smallest and the largest element at l^{th} position which contributes to sum $= s$ and length $= l$.
    for $i = endPt; i \geq startPt; i - -$ **do**
        $maxSum = findMaxSum(n, s, l, i)$
        $minSum = findMinSum(n, s, l, i)$
        if $minSum \leq s \leq maxSum$ **then**
            return i ▷ i is an element
        end if
    end for
end function

6.1.2 Core Idea for Local Search Algorithm

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_{min} and l_{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 6.1 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 $A_{maximal}$.

2. Figure 6.2 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, $A_{minimal}$.

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 the subset is ensured by using lookup technique introduced in Section B.2. This establishes the correctness of the Local Search Algorithms using Maximal and Minimal Subsets.

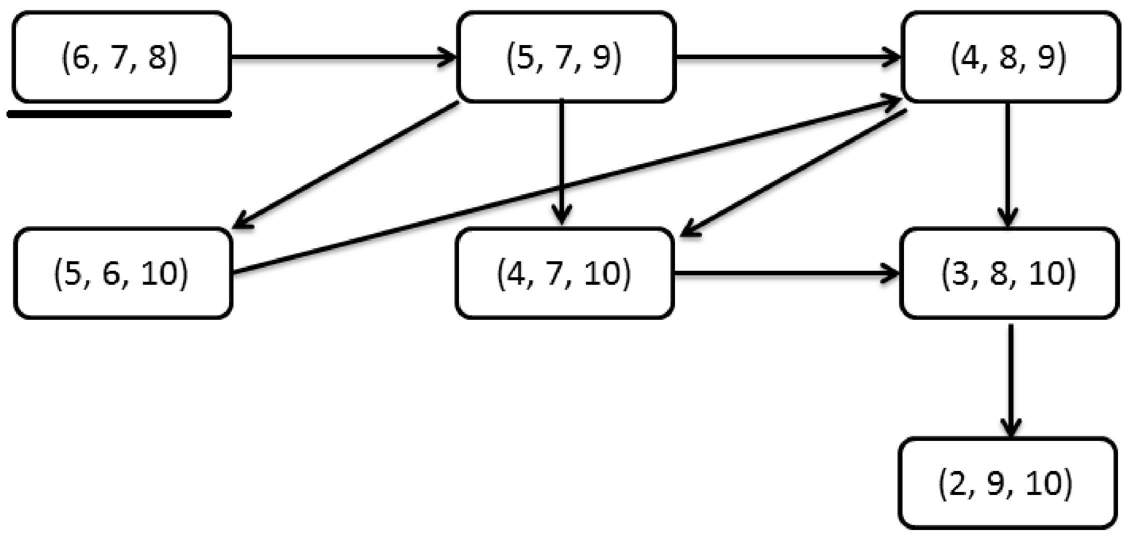


Figure 6.1: Local search for $n = 10$, $sum = 21$ and $length = 3$ with maximal subset as the starting point.

4. Since we know the count of all subsets of X_n with $Sum = S$ and $Length = l$, 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.

6.1.3 Algorithm and Complexities

Local Search using Maximal Subset: Algorithm 17 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, $A_{maximal}$. 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 17, 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 B.2. 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 18 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, $A_{minimal}$. 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 18 uses the same queue data structure and checks the uniqueness among the subsets by using the concept of lookup table as Algorithm 17. This algorithm is terminated when all subsets are generated.

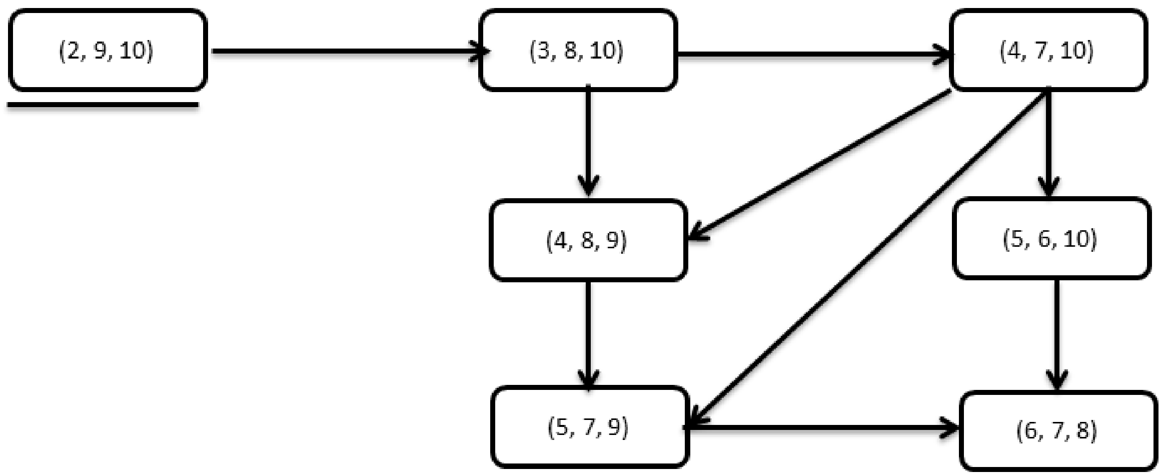


Figure 6.2: Local search for $n = 10$, $sum = 21$ and $length = 3$ with minimal subset as the starting point.

Complexities: Time complexity of these algorithms is complexity of while loop $\times$ complexity of for loop, i.e., $maximum\ no.\ of\ subsets\ *\ length\ of\ each\ subset$. The complexity of the length of each subset variable is $\mathcal{O}(n)$ but the time complexity of $maximum\ no.\ of\ subsets$ variable is exponential. This makes the algorithm exhaustive. Time complexity is $\mathcal{O}(2^n \cdot n^{\frac{-3}{2}} \cdot n) = \mathcal{O}(2^n \cdot n^{\frac{-1}{2}}) = \mathcal{O}(\frac{2^n}{\sqrt{n}})$. The space complexity for these algorithms is equal to the size of storage queue i.e. $maximum\ no.\ of\ subsets * length\ of\ each\ subset$. The time complexity is similar. The complexity of the $Length\ of\ each\ subset$ variable is $\mathcal{O}(n)$ but the space complexity of $maximum\ no.\ of\ subsets$ variable is exponential, $\mathcal{O}(\frac{2^n}{\sqrt{n}})$.

6.2 Summary

In this chapter we have introduced a class of subsets called Maximal or Minimal subset which are determined by dividing power set of X_n on the basis of their sum and length. They are systematically defined in this chapter. We have presented the last alternate enumeration technique for SSP. Local Search is a heuristic algorithm which starts with a seed subset and by following increments and decrements of one unit it generates all desired subsets. Local search can have two starting points: Maximal subset or Minimal subset. We have also presented the algorithms and complexities for Local search algorithm.

Algorithm 17 Local Search for Maximal Subset

```
function LOCALSEARCH(n, s, l)
    maximalSet = MAXIMALSUBSET(n, s, l)
    queue.push(maximalSet)
    allSubsetsGenerated = LD[n][s][l]
    while allSubsetsGenerated > 0 do
        reqSet = queue.pop()
        for i = 1; i ≤ len − 1; i + + do
            if reqSet[i] − 1 > reqSet[i − 1] then
                reqSet[i]− = 1                    ▷ First decrementing the element by 1
                decrement = True
            end if
            for j = i + 1; j ≤ l; j + + do
                if reqSet[j] + 1 < reqSet[j + 1] then
                    reqSet[j]+ = 1
                    increment = True
                end if
                if (reqSet is unique) and (decrement) and (increment) then
                    print reqSet
                    queue.push(reqSet)
                    allSubsetsGenerated − −
                end if
            end for
        end for
    end while
end function
```

Algorithm 18 Local Search for Minimal Subset

```
function LOCALSEARCH(n, s, l)
    minimalSet = MINIMALSUBSET(n, s, l)
    queue.push(minimalSet)
    allSubsetsGenerated = LD[n][s][l]
    while allSubsetsGenerated > 0 do
        reqSet = queue.pop()
        for i = 1; i ≤ len − 1; i + + do
            if reqSet[i] + 1 < reqSet[i + 1] then
                reqSet[i]+ = 1                    ▷ First incrementing the element by 1
                increment = True
            end if
            for j = i + 1; j ≤ l; j + + do
                if reqSet[j] − 1 > reqSet[j − 1] then
                    reqSet[j]− = 1
                    deccrement = True
                end if
                if (reqSet is unique) and (decrement) and (increment) then
                    print reqSet
                    queue.push(reqSet)
                    allSubsetsGenerated − −
                end if
            end for
        end for
    end while
end function
```

Chapter 7

Experimental Results

This chapter presents the experiments that we have conducted to validate the efficiency and effectiveness of all the proposed algorithms: Sum Distribution Generator (SDG), Length Distribution Generator (LDG), Basic Bucket Algorithm (Basic BA), Maximum Frequency Driven Bucket Algorithm (Max FD), Minimum Frequency Driven Bucket Algorithm (Min FD), Local Search Algorithm using Maximal Subset (LS MaxS) and Local Search Algorithm using Minimal Subset (LS MinS).

In Section 7.1, we present a consolidated algorithmic summary of all the proposed alternate enumeration techniques for solving Subset Sum Problem (SSP) along with their time and space complexity. These algorithms were systematically explored in previous chapters. Section 7.2 covers the experimental setup in detail. Section 7.3 presents the ratios of number of excess subsets which were explored while trying to find the exact solution to the overall number of possible subsets for all the algorithms. In Section 7.4, we present the empirical data that was obtained while running the experiments and we highlight all the interesting findings.

7.1 Summary of Alternate Enumeration Techniques

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

Problem Statement: Find all subsets of $\mathcal{P}\left(X_n\right)$ which sum up to S, where X_n is the set of first n natural numbers, $X_n = \{1, 2 \ldots n\}$			
Algorithm	**Core Idea**	**Time Complexity**	**Space Complexity**
Backtracking Algorithm (Benchmark) (section-4.1)	It is an improved and systematic brute force approach for generating various subsets with $Sum = S$. 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-4.2)	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 $Sum = S$ are generated by subsets of X_{n-1} with $Sum = (S - n)$.	$\mathcal{O}(2^n * n^{\frac{3}{2}})$	$\mathcal{O}(2^n * n^{\frac{3}{2}})$
Subset Generator using Length-Sum Distribution (LDG) (section-4.3)	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 $(Sum = S, Length = l)$ are generated by subsets of X_{n-1} with $(Sum = S - n, Length = l - 1)$.	$\mathcal{O}(2^n * n^{\frac{5}{2}})$	$\mathcal{O}(2^n * n^{\frac{5}{2}})$
Basic Bucket Algorithm (Basic BA) (section-5.1)	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.	$\mathcal{O}(2^{2n} \cdot n^{-3})$	$\mathcal{O}(2^n)$
Maximum Frequency Driven Bucket Algorithm (Max FD) (section-5.2)	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 $Sum = S$.	$\mathcal{O}(2^{2n} \cdot n^{-3})$	$\mathcal{O}(2^n)$
Minimum Frequency Driven Bucket Algorithm (Min FD) (section-5.2)	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 $Sum = S$.	$\mathcal{O}(2^{2n} \cdot n^{-3})$	$\mathcal{O}(2^n)$

Local Search using Maximal Subset (LS MaxS) (section-6.1)	This heuristic algorithm finds all 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. Every increment or decrement consists 1 unit.	$\mathcal{O}(\frac{2^n}{\sqrt{n}})$	$\mathcal{O}(\frac{2^n}{\sqrt{n}})$
Local Search using Minimal Subset (LS MinS) (section-6.1)	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 1 unit.	$\mathcal{O}(\frac{2^n}{\sqrt{n}})$	$\mathcal{O}(\frac{2^n}{\sqrt{n}})$

Table 7.1: 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.

7.2 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 7.1, 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 $\mathcal{O}(n)$. 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 $\mathcal{O}(midSum(n))$. This upper bound of the Sum Distribution $SD[n][midSum(n)] = S(n) \approx \sqrt{\frac{6}{\pi}} \cdot 2^n \cdot n^{\frac{-3}{2}}$ is explained in Appendix C. Table 7.2 presents the count of number of subsets X_n with $S = 2n$ and $S = (\frac{n(n+1)}{4} - n)$. This table gives an estimate of the values plotted in Figure 7.1. Figure 7.1(a, c) plot the number of subsets of X_n with $(n \in [1, 250], S = 2n)$ and $(n \in [1, 50], S = (\frac{n(n+1)}{4} - n))$ respectively. Figure 7.1(b, d) plot the log to the base 10 of number of subsets of X_n with $(n \in [1, 250], S = 2n)$ and $(n \in [1, 50], S = (\frac{n(n+1)}{4} - n))$ respectively. Since the values of number of subsets for a particular S increases exponentially with n, we have plotted Figure 7.1(b, d) by using the logarithmic function. This helps in approximating the size of the problem space.

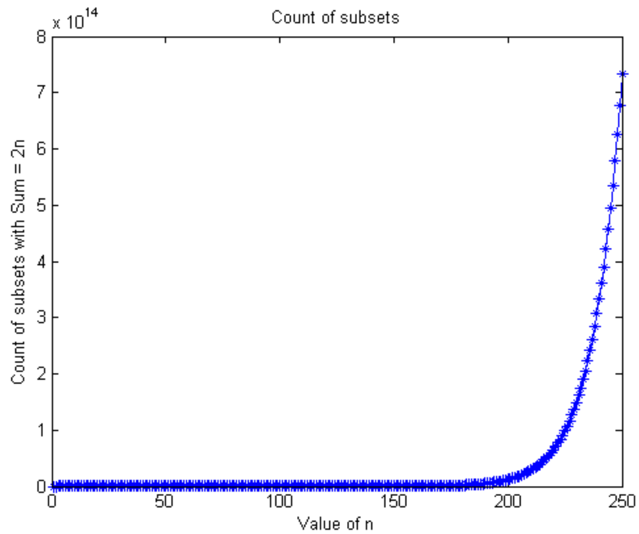
n	Count of Subsets of X_n with $S = 2n$	n	Count of Subsets of X_n with $S = (\frac{n(n+1)}{4} - n)$
6	2	6	2
7	5	7	5
8	8	8	8
9	13	9	13
10	134	10	24
50	416868	15	521
100	482240364	20	11812
150	114613846376	30	7206286
200	11954655830925	40	5076120114
250	732839540340934	50	3831141038816

Table 7.2: Count of number of subsets X_n with $S = 2n$ and $S = (\frac{n(n+1)}{4} - n)$ respectively.

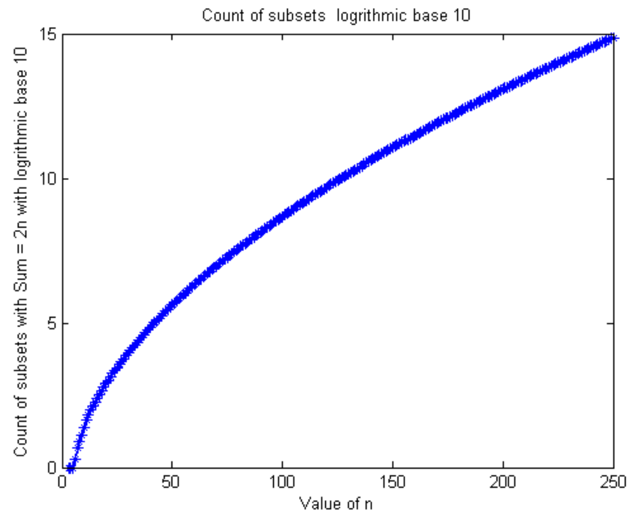
7.3 Excess Subset Generation Analysis

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 is not equal to S.

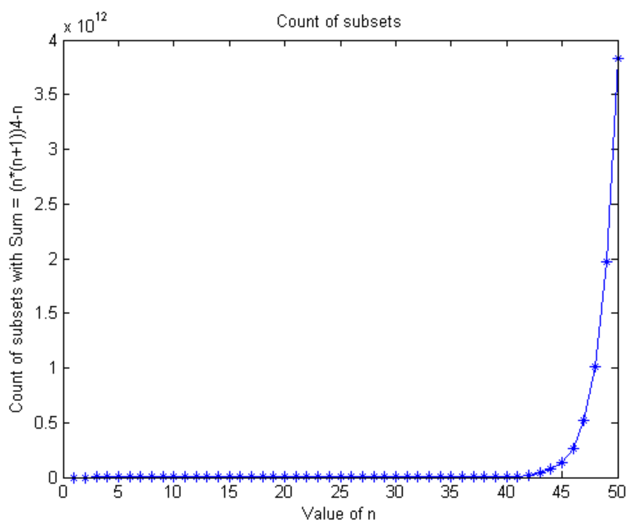
In backtracking method, at every step of subset generation we either include or exclude an element. This creates a recursive tree and a branch is terminated when the current sum exceeds the target. This way we explore more subsets than desired sum. Similarly, in rest of the alternate enumeration techniques 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 7.3 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



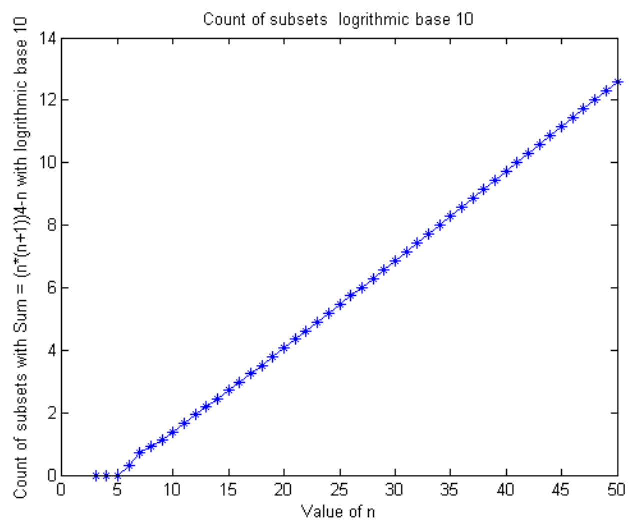
(a) Plot of number of subset of X_n at $S = 2n$ vs n



(b) Plot of log_{10}(number of subset of X_n) at $2n$ vs n



(c) Plot of number of subset of X_n at $S = (\frac{n(n+1)}{4} - n)$ vs n



(d) Plot of log_{10}(number of subset of X_n) at $S = (\frac{n(n+1)}{4} - n)$ vs n

Figure 7.1: 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 $S = (\frac{n(n+1)}{4} - n)$ and plot graph for n varying from 1 to 50. (c) Figure represents graph for number of subsets of X_n with $S = (\frac{n(n+1)}{4} - n)$ where $n \in [1, 50]$. (d) Figure represents graph for number of subsets of X_n with $S = (\frac{n(n+1)}{4} - n)$ with logarithmic base 10 where $n \in [1, 50]$.

ratio of explored subsets to the number of subsets in the final solution for all eight alternate enumeration techniques.

Following observations can be made based on the data presented in Table 7.3:

1. For a given value of n and S, desired ratio is the 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 benchmark 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: $\frac{737}{67} = 11$.

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 than the desired ratio. This observation and the correctness of these algorithms ensure the completeness of the results.

4. Backtracking (benchmark) algorithm explores most number of subsets in order to generate the desired subsets. Backtracking 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 LSMinS are heuristic algorithms, they explore lesser number of subsets compared to backtracking 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 $\frac{93}{67} = 1.3881$ and $\frac{103}{67} = 1.5373$.

   (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 7.4.

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 $\frac{150}{67} = 2.2388$.

(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 7.10 of Section 7.4.

(d) SDG explores more subsets than LDG but it performs better than backtracking algorithm. While backtracking 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 Chapter 2.

(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 Min FD 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 backtracking (benchmark) but can not compute all subsets for a considerable value of n and S within short amount of time.

n	S	$\|Subsets\|$	Bench mark	SDG	LDG	Basic BA	Max FD	Min FD	LS MaxS	LS MinS
12	24	67	11	3.1940	2.2388	5.0896	2.4776	3.5970	**1.3881**	1.5373
12	27	84	20.5952	2.7857	2.0952	5.1548	2.6190	4.2262	1.3690	**1.2976**
15	30	186	21.4194	6.2097	1.6882	4.7742	2.3871	4	**1.1882**	1.2903
15	45	521	23.3282	2.8177	1.3013	-	2.8503	1.3129	1.0211	**1.0058**
16	32	253	60.6087	8.2806	1.6719	5.4664	2.3478	3.7470	**1.1621**	1.2332
17	59	1764	27.0947	2.9127	1.3622	-	2.9892	-	**1.0176**	1.1037
20	40	806	73.6390	30.0447	1.8189	-	2.2667	-	**1.0707**	1.1191
20	85	11812	28.4236	2.9261	1.6258	-	-	-	1.2332	**1.2281**
22	104	41552	34.7394	2.9706	**1.8080**	-	-	-	-	-

Table 7.3: 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.

7.4 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, $Sum(A)$ belonging to the range $[0, maxSum(n)] = [0, \frac{n(n+1)}{2}]$ where $A \in \mathcal{P}(X_n)$. Choosing different values of sum between 0 to $maxSum(n)$ is the core idea behind these experiments. Table 7.4 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 $Sum = S_1$. 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 7.5: Time taken (in seconds) by LS MaxS and LS MinS in CA-SSR. Table 7.6: Time taken (in seconds) by Basic BA, Max FD and Min FD in CA-SSR.
CA-LSR $[midSum(n) - n, midSum(n)]$	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 $Sum = S_2$. For every values of n, this larger range varies from $midSum(n) - n$ to $midSum(n)$ i.e. $\forall n\ S_2 \in [midSum(n) - n, midSum(n)]$.	Basic BA, Max FD, Min FD, LS MaxS, LS MinS	Table 7.7: Time taken (in seconds) by Basic BA, Max FD and Min FD in CA-LSR. Table 7.8: Time taken (in seconds) by LS MaxS and LS MinS in CA-LSR.
CA-FSV	In this experiment instead of choosing random vales 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 $S_1 = 2 * n$ and $S_2 = midSum(n) - n$	Basic BA, Max FD, Min FD, LS MaxS, LS MinS, LDG, SDG	Table 7.9: presents the time taken by Max FD, Min FD, Basic BA, LS MaxS, LS MinS, LDG and SDG algorithms where $S_1 = 2 * n$ and $S_2 = midSum(n) - n$

CA-SLN	For this experiment instead of fixing the value of sum S, we vary S from 0 to $maxSum(n) = \frac{n(n+1)}{2}$. This experiment helps us in analyzing the performance of SDG and LDG algorithms against Benchmark (backtracking) algorithm. In this experiment we enumerate all 2^n subsets of X_n	SDG, LDG, Benchmark	Table 7.10: presents the comparison of SDG and LDG with 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 7.2: Plot of SDG, LDG and Benchmark 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 7.4: 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 7.3, 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 performance 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 Benchmark (CA-SLN) we compare LDG, SDG with benchmark to show that our alternate enumeration techniques performs better than the existing algorithms. Using nave 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					
n	S	Time(in sec)	n	S	Time(in sec)
3	1	0.00015	24	47	19.862
4	1	0.00012	25	36	2.0454
5	1	0.00015	26	10	0.0023
6	3	0.00024	27	17	0.0114
7	4	0.00023	28	5	0.002251
8	10	0.00098	29	8	0.002551
9	5	0.00039	30	8	0.00289
10	16	0.0037	31	53	168.747
11	20	0.0094	32	58	510.344
12	11	0.0018	33	31	1.00335
13	6	0.00070	34	56	411.957
14	1	0.00060	35	50	124.164
15	17	0.01081	36	47	67.4379
16	4	0.00086	37	62	1748.339
17	32	0.30375	38	74	18096.70
18	17	0.00682	39	56	588.9686
19	33	0.46984	40	58	951.2177
20	10	0.00144	41	32	1.890367
21	28	0.19325	42	55	561.6479
22	14	0.00403	43	46	76.02470
23	21	0.03176	44	44	48.61702

Time taken(in sec) by LS MinS in					
n	S	Time(in sec)	n	S	Time(in sec)
3	1	0.00020	24	47	21.2260
4	1	0.00019	25	19	0.01893
5	1	0.00020	26	52	77.8854
6	2	0.00022	27	22	0.05621
7	7	0.00073	28	10	0.00278
8	6	0.00049	29	47	41.2712
9	9	0.00092	30	58	434.903
10	4	0.00047	31	10	0.00344
11	22	0.01392	32	12	0.00514
12	9	0.00122	33	10	0.00396
13	10	0.00145	34	27	0.356276
14	3	0.00069	35	43	26.97922
15	15	0.00634	36	44	36.25960
16	26	0.09706	37	32	1.74147
17	20	0.02005	38	34	3.16681
18	11	0.00148	39	59	1143.04
19	28	0.15588	40	13	0.00915
20	10	0.00149	41	55	556.808
21	31	0.43241	42	26	0.35234
22	28	0.22006	43	14	0.01238
23	20	0.02313	44	40	18.9293

Table 7.5: 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 $\forall n,\ S_1 \in [1, 2n]$.

n	S	Time taken(in sec) by Basic BA in CA-SSR
3	1	0.000551
4	1	0.0004.20
5	2	0.000138
6	3	0.000247
7	5	0.000405
8	2	0.000849
9	2	0.000885
10	16	0.05103
11	13	0.01842
12	13	0.02192
13	9	0.00265
14	16	0.08399
15	26	7.03249
16	31	60.6872
17	34	241.571
18	6	0.00106
19	19	0.75584
20	36	1261.39
21	15	0.09077
22	41	12918.6

n	S	Time taken(in sec) by Max FD in CA-SSR
3	1	0.00076
4	1	0.000564
5	1	0.000569
6	3	0.001332
7	6	0.003052
8	10	0.000837
9	12	0.00139
10	7	0.00384
11	20	0.12669
12	24	0.19757
13	24	0.220844
14	5	0.0011010
15	2	0.0003778
16	9	0.0040118
17	11	0.007174
18	7	0.0024759
19	3	0.0008509
20	33	4.143428
21	11	0.007366
22	37	11.79708
23	29	1.868481
24	42	42.95121
25	22	0.274363
26	4	0.0009
27	5	0.001708
28	29	2.35588
29	27	1.51263
30	13	0.029837
31	15	0.068043
32	17	0.113950
33	57	1822.731
34	47	237.329
35	36	21.1548
36	71	32840.56

n	S	Time taken(in sec) by Min FD in CA-SSR
3	1	0.00094
4	1	0.00065
5	2	0.00072
6	4	0.00156
7	1	0.00073
8	1	0.00073
9	3	0.00016
10	3	0.00172
11	7	0.00503
12	24	0.29195
13	21	0.14098
14	10	0.00586
15	23	0.25498
16	22	0.21270
17	34	4.35665
18	15	0.04607
19	20	0.15957
20	15	0.03706
21	9	0.00526
22	6	0.00236
23	34	8.08626
24	42	52.7242
25	9	0.00573
26	29	2.61190
27	33	8.01348
28	13	0.03223
29	51	523.152
30	24	0.73111
31	33	9.51999
32	41	71.6738
33	43	117.805
34	23	0.71141
35	14	0.08665
36	70	33113.13

Table 7.6: 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-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

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	57	793.294

Table 7.7: 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 $\forall n,\ S_2 \in [midSum(n) - n, midSum(n)]$.

Time taken(in sec) by LS MaxS in CA-LSR					
n	S	Time(in sec)	n	S	Time(in sec)
3	1	0.00003	13	39	0.12096
4	2	0.00025	14	49	0.34326
5	5	0.00057	15	47	0.77065
6	6	0.00049	16	54	2.65762
7	8	0.00067	17	72	10.4998
8	18	0.001429	18	67	30.6323
9	20	0.003262	19	87	149.328
10	18	0.00491	20	97	649.048
11	23	0.01419	21	94	1635.28
12	30	0.04604	22	114	8633.37

Time taken(in sec) by LS MinS in CA-LSR					
n	S	Time(in sec)	n	S	Time(in sec)
3	1	0.00062	13	41	0.12125
4	2	0.00374	14	50	0.33774
5	6	0.00473	15	56	0.87691
6	7	0.01077	16	56	2.99347
7	11	0.00115	17	66	11.1003
8	13	0.0102	18	68	32.6975
9	14	0.00251	19	92	124.407
10	23	0.00898	20	100	639.238
11	29	0.02194	21	114	1881.25
12	31	0.05642	22	116	8897.909

Table 7.8: 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 $\forall n,\ S_2 \in [midSum(n) - n, midSum(n)]$.

n	S	$\|Subsets\|$	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	0.009596
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	0.03134
15	30	186	1.381	7.641	21.208	0.11	0.133	0.00499	0.014521
15	45	521	14.031	353.746	1388.5	0.798	0.955	0.00472	0.056801
16	32	253	2.341	11.761	77.48	0.211	0.255	0.00615	0.017711
16	52	965	45.83	1224.328	-	2.882	3.394	0.0103	0.11678
17	34	343	4.414	27.817	236.119	0.412	0.501	0.00821	0.024524
17	59	1764	153.31	-	-	10.144	12.058	0.01556	0.233748
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	0.03646	0.473109
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	2.08991
21	42	1055	38.779	268.505	-	5.177	6.298	0.0664	0.072871
21	94	21985	25300.63	-	-	2084.648	2421.476	0.22368	4.134605
22	44	1369	64.492	842.423	-	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	-	-	155.258	190.273	1.06806	0.352428
27	162	1038222	-	-	-	-	-	10.77463	345.7571
30	60	9141	-	-	-	733.963	897.121	1.921185	-
30	202	7206286	-	-	-	-	-	98.64595	-

Table 7.9: 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 $S_2 = midSum(n) - n$. This table shows Time taken (in seconds) by all alternate techniques for calculating for these pairs.

n	$\|Subsets\|$	LDG	SDG	Benchmark
2	4	0.00030	0.00018	0.0008
3	8	0.00024	0.00013	0.0012
4	16	0.00032	0.00015	0.0014
5	32	0.00037	0.00017	0.0026

6	64	0.00053	0.00023	0.0054
7	128	0.00061	0.00027	0.0116
8	256	0.00099	0.00032	0.0218
9	512	0.00122	0.00039	0.0423
10	1024	0.00218	0.00052	0.0854
11	2048	0.00257	0.00072	0.2137
12	4096	0.00391	0.00103	0.4542
13	8192	0.00532	0.00174	0.8522
14	16384	0.00863	0.00272	1.5655
15	32768	0.01223	0.00483	3.5153
16	65536	0.02119	0.00927	6.1746
17	131072	0.03060	0.01531	12.9676
18	262144	0.05147	0.02669	24.3167
19	524288	0.07002	0.05230	44.9257
20	1048576	0.12088	0.10512	92.8140
21	2097152	0.21260	0.20231	170.9037
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	12.60573	-
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	-

Table 7.10: Comparison of SDG and LDG with Benchmark 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.

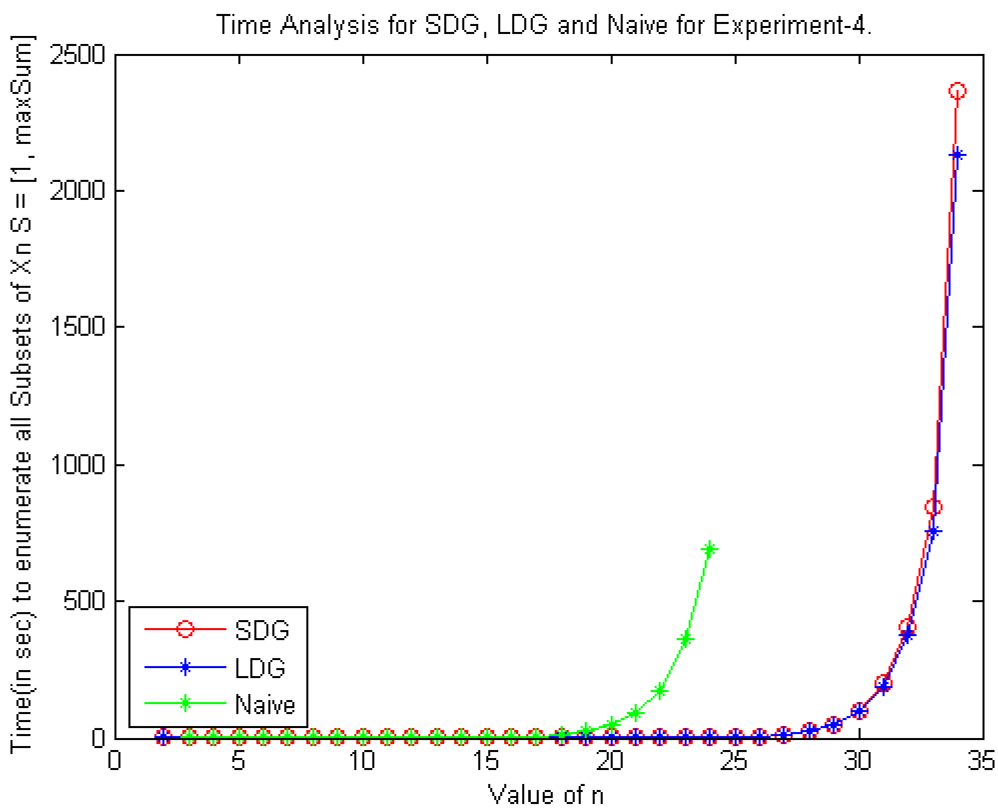


Figure 7.2: Plot of SDG, LDG and Benchmark algorithm while enumerating all 2^n subsets of X_n for every value of sum S in range $[0, \frac{n(n+1)}{2}]$. This graph plots time taken by these algorithms to enumerate each and every subset in CA-SLN.

Chapter 8

Conclusion and Future Work

8.1 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 $\mathcal{P}\left(X_n\right)$ 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 (Benchmark) algorithm. It is an improved and systematic brute force approach for generating various subsets with $Sum = S$. 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 $Sum = S$. Time and space complexities for this algorithm are $\mathcal{O}(n \times 2^n)$ and $\mathcal{O}(n)$ 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 2.1. This algorithm is executed using dynamic programming. Subsets of X_n with $Sum = S$ are generated by subsets of X_{n-1} with $Sum = S$ and $Sum = (S - n)$. Time and space complexities for this algorithm are $\mathcal{O}(2^n * n^{\frac{3}{2}})$ and $\mathcal{O}(2^n * n^{\frac{3}{2}})$ 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 2.23. This algorithm is executed using dynamic programming. Subsets of X_n with $(Sum = S, Length = l)$ are generated by subsets of X_{n-1} with $(Sum = S, Length = l)$ and $(Sum = S - n, Length = l - 1)$. Time and space complexities for this algorithm are $\mathcal{O}(2^n * n^{\frac{5}{2}})$ and $\mathcal{O}(2^n * n^{\frac{5}{2}})$ 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 B and ensures uniqueness among and within the subsets. Time and space complexities for this algorithm are $\mathcal{O}(2^{2n} \cdot n^{-3})$ and $\mathcal{O}(2^n)$ 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 $Sum = S$. For Min FD we select maximum element with minimum frequency to generate all $SD[n][S]$ number of subsets of X_n with $Sum = S$. These methods use the concept of lookup table explained in Section B and ensure uniqueness among and within the subsets. Time and space complexities for this algorithm are $\mathcal{O}(2^{2n} \cdot n^{-3})$ and $\mathcal{O}(2^n)$ 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 $\mathcal{O}(\frac{2^n}{\sqrt{n}})$ and $\mathcal{O}(\frac{2^n}{\sqrt{n}})$ respectively.

Conjecture 8.1.1. *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 $\mathcal{O}(SD[n][S])$ complexity. That is, for no case would it be impossible to get all solutions of Subset Sum Problem if we only enumerate $(c \cdot SD[n][S])$ subsets where c is a constant.*

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

8.2 Future Work

In our future work, we extend the idea of enumerating Subset Sum Problem by using a new data structure using an efficient data structure Descending Sumset Tree i.e. DST. DST is a tree-like data structure which stores all the elements of X_n in descending order. The various traversal techniques of the tree spawns all this desired subsets of X_n. There are two ways of storing elements in a Sumset Tree: Ascending or Descending. Either one can be extended as part of future work.

Apart from root node, DST has two types of nodes: $EvenNode$ and $OddNode$. $EvenNode$ contains different elements and $OddNode$ contains ϕ. Figure 8.1 represents a basic structure of DST. From this figure, we can understand the structure of DST.

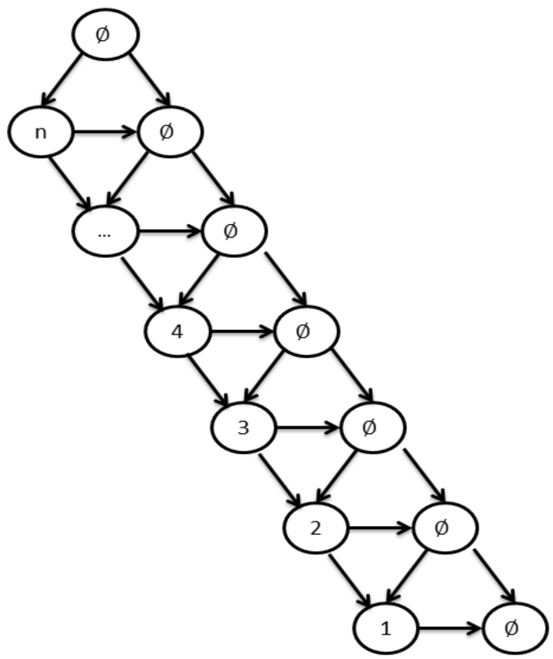


Figure 8.1: Basic structure of Descending Sumset Tree (DST)

DST can be used to overcome the shortcomings of the backtracking algorithm with the help of *Descending Sumset Tree (DST)*. As we can see from Figure 8.1, DST is designed in a such a way that it eliminates the possibility of backtracking. All the paths between $root$ node and the last $OddNode$ of DST will create the power set of X_n, where the elements of X_n are stored in this DST.

Using DST we can develop alternate enumeration techniques for solving SSP. These can be different traversal algorithms using properties like *sum* of a subset, *length* of a subset etc.

1. A naive simple tree traversal can enumerate all 2^n subsets.

2. A traversal algorithm with constraints on target sum can produce subsets of X_n with $Sum = S$.

3. A traversal algorithm with constraints on target length can produce l length subsets of X_n.

4. A traversal algorithm with constraints on target sum and target length can produce subsets of X_n with $Sum = S$ and $Length = l$.

Figure 8.2(a) presents the traversal path which generates the singleton subset $\{6\}$ of X_6. The traversal generates the subset $\{\phi, 6, \phi, \phi, \phi, \phi, \phi\} \equiv \{6\}$. Figure 8.2(b, c) present the traversal path which generates the subsets $\{4, 2\}$ and $\{3, 2, 1\}$ of X_6 respectively. These subsets add up to the sum of 6.

This work has a lot of potential but current algorithms high theoretical complexity. The improvement can be made by proposing and designing more enumeration techniques using DST.

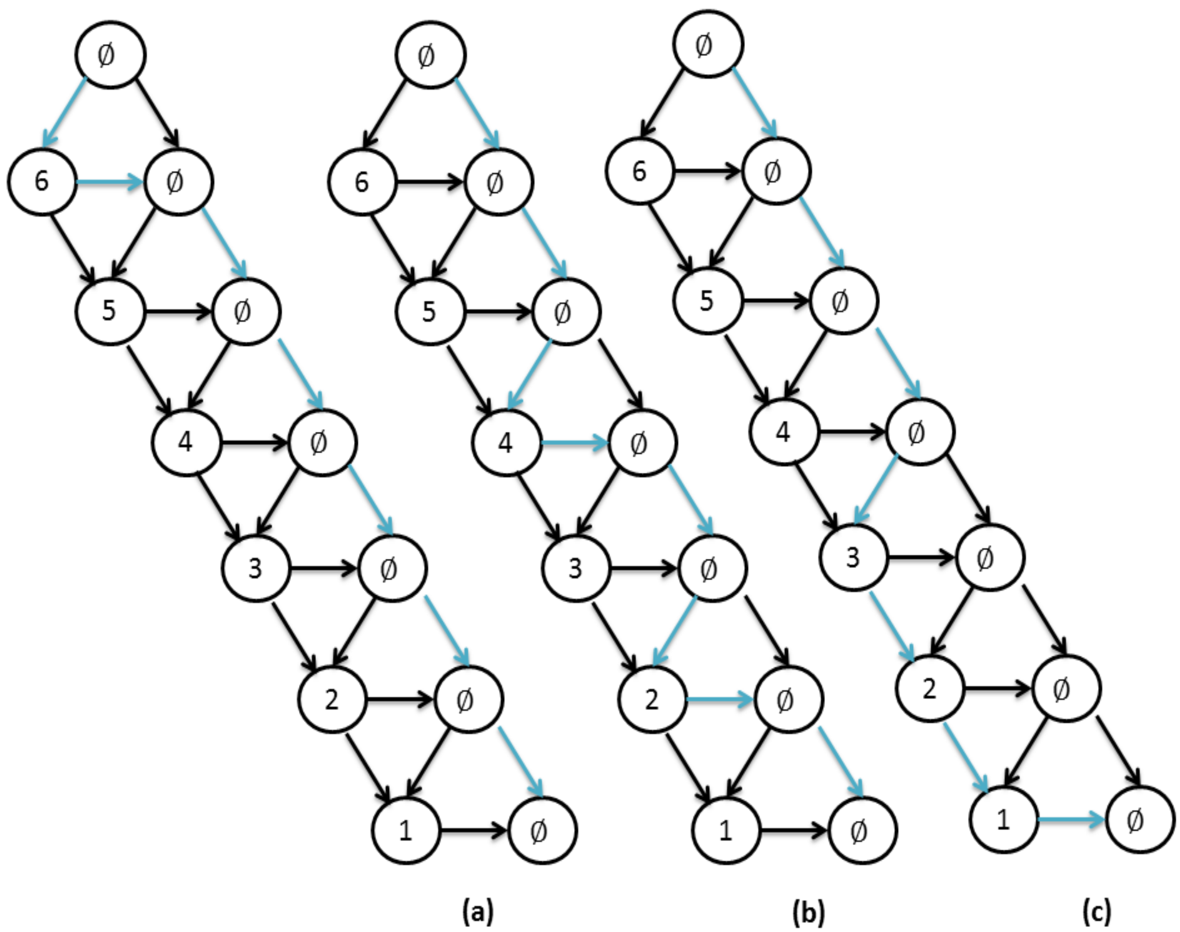


Figure 8.2: Various examples of Descending Sumset Tree(DST) which stores elements of X_6. Subsets $\{\{6\}, \{4, 2\}, \{3, 2, 1\}\}$ are generated by using a simple tree traversal. These generations are shown in (a), (b) and (c) respectively.

Apart from DST this thesis work can be extended in following ways:

- 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.

- 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.

- 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.

Appendix A

Glossary

X_n	First n natural numbers. $X_n = \{1, 2 \dots n\}$, where n is a positive integer.
U	Universal set of X_n. $U = \{1, 2 \dots n\}$
$P(X_n)$	Power set of X_n, is set of all subsets of X_n including empty set ϕ and U or X_n itself.
$\|P(X_n)\|$	Cardinality of $P(X_n) = 2^n$
$maxSum(n)$	Sum of all the elements of U, $maxSum(n) = \frac{n(n+1)}{2}$.
$Sum(A)$	Sum of all elements of set A. $Sum(A) \in [0, \frac{n(n+1)}{2}]$
$Len(A)$	Length of set $A = Len(A) = \|A\|$. $Len(A) \in [0, l]$
$Subset_{(min,l)}$	Subset of X_n with minimum sum and length $= l$. $Subset_{(min,l)} = \{1, 2 \dots l\}$
$minSum(n, l)$	$Sum(Subset_{(min,l)}) = \frac{l(l+1)}{2}$
$Subset_{(max,l)}$	Subset of X_n with maximum sum and length $= l$. $Subset_{(max,l)} = \{n, n-1, n-2 \dots n-(l-1)\}$
$maxSum(n, l)$	$Sum(Subset_{(max,l)}) = \frac{l(2n-l+1)}{2}$
$sum_{(i,j)}$	Set of all the subsets of X_i which add up to a certain sum j, where $i \in [1, n]$ and $\forall j \in [0, n(n+1)/2$.
$SD[i][j]$	The number of subsets of X_i which add up to a certain sum j, where $i \in [1, n]$ and $\forall j \in [0, n(n+1)/2$. $SD[i][j]$ is called the sum distribution.
$length_{(i,j,k)}$	Set of all the subsets of X_i with $Sum = j$ and $length = k$, where $i, k \in [1, n]$ and $\forall k \in [0, n(n+1)/2$.
$LD[i][j][k]$	The number of subsets of X_i with $Sum = j$ and $length = k$, where $i, k \in [1, n]$ and $\forall k \in [0, n(n+1)/2$. $LD[i][j][k]$ is called the length distribution.

$element_{(i,j,k)}$	A class of subsets with all the subsets of $P(X_i)$ which add up to a sum of j and contain an element k, where $i, k \in [1, n]$ and $\forall j \in [0, n(n+1)/2$.
$ED[i][j][k]$	The count of element k in all the subsets of X_i which add up to a certain sum j, where $i, k \in [1, n]$ and $\forall j \in [0, n(n+1)/2$. $ED[i][j][k]$ is called the element distribution.

Appendix B

Lookup Technique

B.1 Numbering of subsets

In this thesis, it is very imperative for us to number all the power sets of X_n. This provides an easier way to explain the enumeration techniques and related calculations.

As described in Section 1.3, the cardinality of the set $\{\mathcal{P}(X_n) \cup \phi\}$ is 2^n, $|\{\mathcal{P}(X_n) \cup \phi\}| = 2^n$. Since each subset is unique or in other words, group of elements of each subset is different than others, each subset has one-to-one mapping to all integers between 0 to 2^n. For example, ϕ is mapped to 0 and universal set($\{1, 2, 3 \ldots n\}$) is mapped to $2^n - 1$. For any subset A belonging to $\mathcal{P}(X_n)$, $A \in \mathcal{P}(X_n)$, $A = \{A_1, A_2 \ldots A_l\}$, l is the cardinality of subset A, $l = |A|$ the numbering of subset A among all the power set, num is defined as follows:

$$S_{num} = \sum_{i=1}^{l} 2^{A_i - 1} \tag{B.1}$$

Table B.1 represents the positive integer against every subset of $\mathcal{P}(X_0)$, $\mathcal{P}(X_1)$, $\mathcal{P}(X_2)$, $\mathcal{P}(X_3)$ and $\mathcal{P}(X_4)$ respectively. This numbering technique is the fundamental concept of the lookup table illustrated in next section, Section B.2.

B.2 Lookup Table

The main concept behind numbering of subsets is introduced in Section B.1, is the mapping of each subset with a unique integer. The same concept is 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 to the corresponding integer, denoted by num and defined in Equation B.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 $\mathcal{P}(X_n)$.

Values of num, $n=0$	Subset
S_0	$\{\phi\}$

Values of num, $n=1$	Subset
S_0	$\{\phi\}$
S_1	$\{1\}$

Values of num, $n=2$	Subset
S_0	$\{\phi\}$
S_1	$\{1\}$
S_2	$\{2\}$
S_3	$\{1,2\}$

Values of num, $n=3$	Subset
S_0	$\{\phi\}$
S_1	$\{1\}$
S_2	$\{2\}$
S_3	$\{1,2\}$
S_4	$\{3\}$
S_5	$\{1,3\}$
S_6	$\{2,3\}$
S_7	$\{1,2,3\}$

Values of Num	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
$Subset$	ϕ	$\{1\}$	$\{2\}$	$\{1,2\}$	$\{3\}$	$\{1,3\}$	$\{2,3\}$	$\{1,2,3\}$
Values of Num	S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}	S_{14}	S_{15}
$Subset$	$\{4\}$	$\{1,4\}$	$\{2,4\}$	$\{1,2,4\}$	$\{3,4\}$	$\{1,3,4\}$	$\{2,3,4\}$	$\{1,2,3,4\}$

Table B.1: Values of num for subsets of $\mathcal{P}(X_0)$, $\mathcal{P}(X_1)$, $\mathcal{P}(X_2)$, $\mathcal{P}(X_3)$ and $\mathcal{P}(X_4)$ respectively.

The bucket algorithms which are defined in Section 5.1 of Chapter 4 and the local search algorithms which are defined in Section 6.1 of Chapter 6 extensively use the lookup table. Since, in both these algorithms, we iteratively create the subsets, therefore, we require uniqueness amongst these subsets. For each subset, we maintain a $checker$ which is nothing but the value of num corresponding to every partial and full subset of X_n.

Algorithm 19 presents a procedure to maintain lookup table at each step when an element e is updated in a subset A. Element e is added to subset A only if the procedure in Algorithm 19 returns a positive result. First, we define a temporary checker $temp\_checker$, to store the current hash value of the bucket b. Condition in Line 3 checks whether the element e is already present in A or not. Since $checker(b)$ represents a bit vector element e. It is present in subset A, if e^{th} bit (from the left) of $checker(A)$ is set. Similarly, by using concept of bit vectors, Line 7 checks the uniqueness of the resulting subset on adding element e. Element(e) is set only if the resulting subset is unique. Line 11 updates the value of the hash value of subset A by updating $Hash[checker(A)]$. This line successfully adds the element e in subset A.

For every element in subset A, we set the corresponding bit in $checker(A)$. We can store state of each bucket in a reduced format this way. Algorithm 20 is a helper function which returns the value of $checker(A)$ which is converted to corresponding array of elements. In Line 4 to Line 10, we take every bit of $checker(A)$ and find the set positions which are nothing but the elements present in subset A.

Algorithm 19 Lookup Table(element e, subset A)

Adding element e to subset A
$temp\_checker = Hash[checker(A)]$
if $(temp\_checker \ \& \ (1 << e)) > 0$ **then**
    $Return$ ▷ element e is already present in the subset
end if
$temp\_checker| = (1 << e)$
if $Hash[temp\_checker]$ NOT empty **then**
    $Return$ ▷ subset A already exists.
end if
$checker(A) = temp\_checker$
$Return\ Hash[checker(A)] \rightarrow A$

Algorithm 20 BitVectorToSubset(subset A)

generating elements of subset(A) from a bit vector
$elements = [\,]$
$position = 1$
while $checker(A) \neq 0$ **do**
    if $(checker(A)\&1) \neq 0$ **then**
        $elements.add(position)$
    end if
    $position + +$
    $checker = checker >> 1$
end while
$Return\ elements$

Appendix C

Upper Bound on Sum Distribution

In this section, we use definitions and formulas presented in 1.3. By using the maximum limit on the number of subsets with a particular sum, we find an upper bound of our problem.

Sum distribution $SD[n]$, defined in Section 2.1 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 2.1). The maximum value of $SD[n]$ is found at $midSum(n) = \lfloor \frac{n(n+1)}{4} \rfloor$. Table C.1 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	1	2	2	3	5	8	14	23	40	70	124	221	397	722

Table C.1: Values of $SD[n][midSum(n)]$ for first 15 natural numbers

For each n, value of $SD[n][midSum(n)]$ presented in table C.1 is the coefficient of $x^{\frac{n(n+1)}{4}}$ in the expansion of $\{(1 + x)(1 + x^2)(1 + x^3) \dots (1 + x^n)\}$. This coefficient is denoted as $S(n)$ and $S(n) \approx \sqrt{\frac{6}{\pi}} \cdot 2^n \cdot n^{\frac{-3}{2}}$ [23]. Therefore, value of maximum number of subsets with sum as $midSum(n)$ has exponential bound, $\mathcal{O}(2^n \cdot n^{\frac{-3}{2}})$. This result is vastly used throughout the thesis in order to find complexities of various enumeration techniques.

Bibliography

[1] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. Introduction to Algorithms. The MIT Press, 3rd edition, 2014.

[2] O. H. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. J. ACM, 22(4):463468, Oct. 1975.

[3] P. Austrin, P. Kaski, M. Koivisto, and J. Määttä. Spacetime tradeos for subset sum: An improved worst case algorithm. In F. Fomin, R. Freivalds, M. Kwiatkowska, and D. Peleg, editors, Automata, Languages, and Programming, volume 7965 of Lecture Notes in Computer Science, pages 4556. Springer Berlin Heidelberg, 2013.

[4] Wikipedia contributors. "Subset sum problem." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 29 Aug. 2016. Web. 29 Aug. 2016.

[5] M. Coster, A. Joux, B. LaMacchia, A. Odlyzko, C.-P. Schnorr, and J. Stern. Improved lowdensity subset sum algorithms. computational complexity, 2(2):111128, 1992.

[6] D. Lokshtanov and J. Nederlof. Saving space by algebraization. In Proceedings of the Fortysecond ACM Symposium on Theory of Computing, STOC 10, pages 321330, New York, NY, USA, 2010. ACM

[7] Z. Galil and O. Margalit. An almost linear-time algorithm for the dense subset-sum problem. SIAM J. Comput., 20(6):11571189, Dec. 1991.

[8] M. Chaimovich. New algorithm for dense subset-sum problem. Astrisque, (258):363373, 1999.

[9] D. Pisinger. Linear time algorithms for knapsack problems with bounded weights. Journal of Algorithms, 33(1):1 14, 1999.

[10] K. Koiliaris and C. Xu. A Faster Pseudopolynomial Time Algorithm for Subset Sum. arXiv: 1507.02318v1 [cs.DS], July 2015

[11] L. Trevisan. Pseudorandomness in computer science and in additive combinatorics. In An irregular mind, volume 21 of Bolyai Soc. Math. Stud., pages 619650. Jnos Bolyai Math. Soc., Budapest, 2010.

[12] F. V. Fomin and D. Kratsch. Exact Exponential Algorithms. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010.

[13] E. Horowitz and S. Sahni. Computing partitions with applications to the knapsack problem. J. Assoc. Comput. Mach., 21:277292, 1974.

[14] G.J. Woeginger. Exact algorithms for NP-hard problems: A survey. In Michael Jnger, Gerhard Reinelt, and Giovanni Rinaldi, editors, Combinatorial Optimization, volume 2570 of Lecture Notes in Computer Science, pages 185208. Springer, 2001. 97

[15] R. Beier and B.Vocking. Random knapsack in expected polynomial time. In Proc. 35th ACM STOC, pages 232241, 2003

[16] G. H. Hardy and E.M.Wright An Introduction to the Theory of Numbers, 4th ed. Clarendon Press, Oxford, England, 1959.

[17] P. C. Gilmore, and R.E. Gomory. Multistage cutting stock problems of two and more dimensions. Oper. Res. 13 (1965), 94-120.

[18] P. C. Gilmore and R.E. Gomory. The theory add computation of knapsack functions. Oper. Re8.14 (1966), 10451074.

[19] P. Austrin, P. Kaski, M. Koivisto, and J. Nederlof. Subset Sum in the Absence of Concentration.

[20] T. Tao. The dichotomy between structure and randomness, arithmetic progressions, and the primes. In International Congress of Mathematicians. Vol. I, pages 581608. Eur. Math. Soc., Zurich, 2007.

[21] T. Tao. Structure and randomness in combinatorics. In FOCS, pages 315. IEEE Computer Society, 2007.

[22] T. Tao. Structure and Randomness. American Mathematical Society, Providence, RI, 2008.

[23] Sullivan, B.D., On a Conjecture of Andrica and Tomescu, J.Integer Sequence 16(2013), Article 13.3.1

[24] M. Silvano, T. Paolo (1990). 4 Subset-sum problem. Knapsack problems: Algorithms and computer interpretations. Wiley-Interscience. pp. 105136. ISBN 0-471-92420-2. MR 1086874.

[25] D Pisinger (1999). Linear Time Algorithms for Knapsack Problems with Bounded Weights. Journal of Algorithms, Volume 33, Number 1, October 1999, pp. 114

[26] K. Konstantinos, X. Chao (2015-07-08). A Faster Pseudopolynomial Time Algorithm for Subset Sum. arXiv:1507.02318

[27] G.H. Bradley. Transformation of integer programs to knapsack problems. Discrete Math 1, 1 (May 1971), 29-45.

[28] H. Greenberg and R.L. Hegerich. A branch search algorithm for the knapsack problem. Manage. Sci. 16, 5 (Jan. 1970), 327-332.

[29] P. J. Kolesar. A branch add bound algorithm for the knapsack problem. Manage. Sci. 18 (May 1967), 723-735.

[30] A.D. Flaxman and B. Przydatek. Solving medium-density subset sum problems in expected polynomial time. In Volker Diekert and Bruno Durand, editors, STACS 2005, volume 3404 of Lecture Notes in Computer Science, pages 305314. Springer Berlin Heidelberg, 2005.

[31] G.L. Nemhauser and Z. Ullman. Discrete dynamic programming and capital allocation. Manage. Sci. 15, 9 (May 1969), 494-505.