Computational Aspects of Ordered Integer Partitions with Bounds*

Roland Glück Dominik Köppl

May 10, 2020

Abstract

This paper is dedicated to the counting problem of writing an integer number z as a sum of an ordered sequence of n integers from n given intervals, i.e., counting the number of configurations $(z_1, \ldots, z_n)$ with $z = z_1 + \cdots + z_n$ for $z_i \in [x_i, y_i]$ with integers x_i and y_i and $1 \leq i \leq n$. We show an algorithm computing this number in $\mathcal{O}\left(nz^{\lg n}\right)$ average time, and a data structure computing this number in $\mathcal{O}(n)$ time independently of z. The data structure is constructed in $\mathcal{O}\left(2^n n^3\right)$ time. Its construction algorithm only depends on the intervals $[x_i, y_i]$ $(1 \leq i \leq n)$. This construction algorithm can be parallelized with $\pi = \mathcal{O}\left(n^3\right)$ processors, yielding $\mathcal{O}\left(2^n \frac{n^3}{\pi}\right)$ construction time with high probability.

1 Introduction

The class of balls and urns problems is classic in the field of enumerative combinatorics: Let us consider n *distinguishable* urns with unbounded capacity that are enumerated from one up to n, and z *indistinguishable* balls. A basic question, called *stars and bars* [8], asks to enumerate all possibilities of how to distribute the balls to the urns. Although its answer of $\binom{z+n-1}{z}$ is trivially computable, there is no known polynomial time (with respect to n) algorithm when adding a seemingly simple constraint like an upper bound to each urn's capacity. Unfortunately, a lot of practical questions involve constraints on the urn's size:

- Creating groups of restricted size is a common problem of everyday's life like assigning z employees to n working teams, or z students into n exercise classes.
- In quantum mechanics, the electron configuration assigns z electrons to n atomic orbitals (see e.g., [15, Chapter X]).
- A chained hash table of size n can use a linked list with a restricted maximum size for each bucket.

*Parts of this work have already been presented at the 12th International Symposium on Experimental Algorithms [10].

- Given a finite set U and n total orders on U, the Pareto order [4] induced by the total orders is an irreflexive, transitive and strict order. The order can be visualized as a Hasse diagram. Common problems ask for (a) the number of nodes on a given depth z of the Hasse diagram, and (b) the number of depths that share the same maximal number of nodes (see Section 9).

This article addresses the aforementioned problems by studying the following formal problem: Given n integer intervals $[x_i, y_i]$ with $x_i, y_i \in \mathbb{Z}$ and an integer $z \in \mathbb{Z}$, return the cardinality of the set $\{(z_1, \ldots, z_n) \in \mathbb{Z}^n : z_i \in [x_i, y_i]\ \forall i \in [1, n] \wedge \sum_{i=1}^{n} z_i = z\}$. Translating this problem by $z' =_{df} z - \sum_{i=1}^{n} x_i$ to the problem of enumerating all tuples $(z_1, \ldots, z_n) \in \mathbb{N}_0^n$ with $0 \leq z_i \leq y_i - x_i$ for $0 \leq i \leq n$ and $\sum_{i=1}^{n} z_i = z'$, we can enumerate all solutions in $\mathcal{O}\left(n^{z'} + 1\right)$ time by an exhaustive search (the addition of 1 is for the check whether z' is negative). This solution becomes rapidly unsatisfying when scaling z. In what follows, we provide two different approaches to tackle this problem, namely

- a divide-and-conquer algorithm computing the enumeration in $\mathcal{O}\left(nz^{\lg n}\right)$ average time (Algorithm 1), and
- a data structure computing this number in $\mathcal{O}(n)$ time independently of z. The data structure is constructed on n bounds in $\mathcal{O}\left(2^n n^3\right)$ time. The construction of the data structure can be parallelized with $\pi = \mathcal{O}\left(n^3\right)$ processors, yielding $\mathcal{O}\left(\frac{2^n n^3}{\pi}\right)$ construction time with high probability (Theorem 5.1). Since the construction is independent of z, the data structure is suitable for multiple queries on the same bounds.

Structure of the Paper. We first introduce the problem formally in Section 2 and introduce the notations of this article. Subsequently, we present the divide-and-conquer algorithm in Section 3. Section 4 sketches the basic idea of the main result of this article with an elaborated example. These basic ideas are formalized, followed by a proof of the correctness. The data structure described in Section 4 is analyzed with respect to its construction and query time in Section 5. Section 6 is devoted to a parallel implementation of the algorithm described in Section 4. Practical improvements are considered in Section 7. The theoretical analysis is reviewed in Section 8, which is dedicated to our experimental results. Finally, we summarize our results and give an outlook to future work in Section 9.

Related Work. We quickly survey some known derivates of the classic stars and bars problem: Andrews [1] reviewed some derivatives for the integer partition problem: One of his problems derives from the classic stars and bars problem by keeping the number of urns arbitrary; here the urns may or may be not distinguishable. If the urns are indistinguishable, it is easy to solve the enhanced version that fixes the number of balls of the fullest urn: The number of possible distributions of n balls into some urns, such that the fullest urn contains exactly k balls, is the number of different ways to put n balls into k urns. In another derivate, we take $a_1, \ldots, a_n$ pairwise coprime numbers and restrict the number of balls in the i-th urn to be a multiple of a_i $(1 \leq i \leq n)$.

Answering whether there exists even a solution is NP-hard [3]. Other literature gives answers for (not ordered) partition with all kinds of constraints and some elementary results about ordered partitions without constraints [6, 11, 16]. The distribution after a random assignment was studied in [19]. The urn model is also used for load-balancing preferring putting a ball into the least loaded urn [17].

To the best of our knowledge, we have not found any deep investigation of the problem of computing the number of ordered integer partition with upper bounds. Although this problem is a nearby generalisation of the concept of the composition of a natural number (see e.g. [16]) the problem with the additional constraint of upper bounds seems not to be treated on a broader way till now. Wirsching [21] introduces the setting of our problem, but does not treat the problem algorithmically. Instead, the work focuses on the Lebesgue-convergence of a normalized counting function based on the urns' capacities under the condition that these capacities are a geometrically ascending sequence.

2 Basic Definitions and Properties

We denote the natural numbers $1, 2, 3 \ldots$ with $\mathbb{N}_+$. If we include zero to this set, we write $\mathbb{N}_0$ instead. $\mathbb{Z}$ denotes the set of integer numbers. Intervals of the form $[a, b]$ with $a \leq b \in \mathbb{Z}$ are integer intervals, i.e. $[a, b] = \{a, \ldots, b\}$. With upper case letters I and J we refer to finite sequences of nonnegative (or positive, depending on the context) integers. By $|I|$ we denote the length of such a sequence. Similarly, by $\mathbb{I}$ we denote finite sequences of integer intervals with length $|\mathbb{I}|$. Letters in calligraphic font like $\mathcal{I}$ or $\mathcal{J}$ denote intervals. The restriction of a function P to a certain interval $\mathcal{I}$ is denoted by $P|_{\mathcal{I}}$.

2.1 Definitions

The enhanced problem with upper bounds is formally defined as follows:

Definition 2.1. Given a sequence $I = (i_1, i_2, \ldots, i_n)$ of n natural numbers and an integer z, we call a sequence $J = (j_1, j_2, \ldots, j_n)$ of n non-negative integers a ***partition of z with respect to I*** if the following two requirements are fulfilled:

1. $\forall k \in \{1 \ldots n\} : j_k \leq i_k$, and
2. $\sum_{k=1}^{n} j_k = z$.

We call I the ***upper bounds*** and z the ***target value***.

Problem 2.2. Our goal is to compute the number of distinct bounded partitions of z with respect to I. We denote this number by $\begin{bmatrix} I \\ z \end{bmatrix}$.

For $0 \leq z \leq \min(i_1, \ldots, i_n)$ the problem is the classic stars and bars problem (the solution is $\binom{n+z-1}{z}$), since none of the upper bounds poses a restriction to the distributions of such a small z.

2.2 Reducing the Restriction with Bounded Intervals

A more general constraint restricts both the upper and the *lower* bounds of a partition. This is captured by the next definition:

Definition 2.3. Given a sequence $\mathbb{I} = ([x_{1_l}, y_1], [x_{2_l}, y_2], \ldots, [x_{n_l}, y_n])$ of n intervals of integers[1] and an integer z, we call a sequence $J = (j_1, j_2, \ldots, j_n)$ of n integers a ***bounded partition of z with respect to $\mathbb{I}$*** if the following two requirements are fulfilled:

1. $\forall k \in \{1 \ldots n\} : x_{k_l} \leq j_l \leq y_k$
2. $\sum_{k=1}^{n} j_k = z$

We call $\mathbb{I}$ the ***bound intervals*** and z the ***target value***.

Problem 2.4. The goal remains the same, namely to compute the number of distinct partitions of z with respect to $\mathbb{I}$, which we denote by $\begin{bmatrix} \mathbb{I} \\ z \end{bmatrix}$.

The following lemma shows that the problem of computing the number of bounded partitions can be reduced to the computation of the number of partitions:

Lemma 2.5. Given $\mathbb{I} = ([x_1, y_1], [x_2, y_2], \ldots, [x_n, y_n])$ a sequence of n intervals on $\mathbb{Z}$ and an arbitrary target value z. Let $I =_{df} (y_1 - x_1, y_2 - x_2, \ldots, y_n - x_n)$ be a sequence of non-negative numbers with length n. Further, set $z' =_{df} z - \sum_{i=1}^{n} x_i$. Then the equality $\begin{bmatrix} \mathbb{I} \\ z \end{bmatrix} = \begin{bmatrix} I \\ z' \end{bmatrix}$ holds.

Proof. Let $J = (j_1, j_2, \ldots, j_n)$ be a bounded partition of z with respect to to $\mathbb{I}$. Then the sequence $J' =_{df} (j_1 - x_1, j_2 - x_2, \ldots, j_n - x_n)$ is a partition of z' with respect to I. Conversely, from every partition $K = (k_1, k_2, \ldots, k_n)$ of z' with respect to I we can construct a bounded partition $K' = (k_1 + x_1, k_2 + x_2, \ldots, k_n + x_n)$ of z with respect to $\mathbb{I}$. So there is a bijection between bounded partitions of z with respect to $\mathbb{I}$ and partitions of z' with respect to I which implies the claim. □ □

The reduction from Lemma 2.5 can be executed in $\mathcal{O}(n)$ time.

Example 2.6. Let us assume we have n urns, for which the j-th urn has a minimum and maximum capacity of x_j and y_j ($0 \leq x_j \leq y_j$), for each $1 \leq j \leq n$. Instead of computing directly the number of different distributions of some z balls, we try to satisfy the minimum constraints by putting in the j-th urn x_j balls (for each $1 \leq j \leq n$). If z is too small, there is no possible distribution. Otherwise, we transforme the problem into distributing $z - \sum_{j=1}^{n} x_j$ balls into n urns, where the j-th urn has a minimum and maximum capacity of 0 and $y_j - x_j$, respectively. But this is exactly Problem 2.2.

[1] In particular, we allow negative integer values.

3 A Simple Divide and Conquer Approach

Lemma 3.1 ([21])**.** Let $I = (i_1, i_2, \ldots, i_n)$ be a sequence of non-negative integer values. By construction, the following properties hold:

a) $\begin{bmatrix} I \\ z \end{bmatrix} = 0$ if $z < 0$ or $z > \sum_{l=1}^{n} i_l$.

b) $\begin{bmatrix} I \\ 0 \end{bmatrix} = \begin{bmatrix} I \\ \sum_{l=1}^{n} i_l \end{bmatrix} = 1$.

c) In the case $I = (i_1)$ we yield $\begin{bmatrix} (i_1) \\ z \end{bmatrix} = \begin{cases} 1 & \text{if } 0 \leq z \leq i_1, \text{ and} \\ 0 & \textit{otherwise.} \end{cases}$

A simple, computational property of Problem 2.2 is the following recursion formula:

Lemma 3.2. For every m with $1 \leq m \leq n-1$ the equality

$$\begin{bmatrix} (i_1, i_2, \ldots, i_n) \\ z \end{bmatrix} = \sum_{k=0}^{z} \begin{bmatrix} (i_1, i_2, \ldots, i_{n-m}) \\ k \end{bmatrix} \cdot \begin{bmatrix} (i_{n-m+1}, i_{n-m+2}, \ldots, i_n) \\ z-k \end{bmatrix}$$

holds.

Proof. For every partition $J = (j_1, j_2, \ldots, j_n)$ of z with respect to $(i_1, i_2, \ldots, i_n)$ we have $a =_{df} \sum_{l=1}^{n-m} j_l = z - \sum_{l=n-m+1}^{n} j_l =_{df} z - b$. Since $0 \leq a, b \leq z$ holds, the claim follows by elementary counting principles. □ □

Another property we exploit is the fact that $z \mapsto \begin{bmatrix} I \\ z \end{bmatrix}$ is symmetric with respect to the half of the sum of the upper bounds.

Lemma 3.3. Given a sequence $I = (i_1, i_2, \ldots, i_n)$ from $\mathbb{N}_0$ and an integer $z \in \mathbb{Z}$, we have the equality $\begin{bmatrix} I \\ z \end{bmatrix} = \begin{bmatrix} I \\ \sum_{k=1}^{n} i_k - z \end{bmatrix}$.

Proof. Let $(j_1, j_2, \ldots, j_n)$ be a partition of z with respect to I. Then the sequence $(i_1 - j_1, i_2 - j_2, \ldots, i_n - j_n)$ is a partition of $\sum_{l=1}^{n} i_l - z$ with respect to I. This means that the partitions of z with respect to I and the partitions of $\sum_{l=1}^{n} i_l - z$ with respect to I are *bijective*, which implies the claim. □ □

Example 3.4. Instead of arguing about how many balls are put into each urn, we argue about how much space is left in each urn after a distribution.

Lemma 3.1 and Lemma 3.3 imply that a solver for Problem 2.2 has to consider only the query values $0 \leq z \leq \left\lfloor \frac{\sum_{k=1}^{n} i_k}{2} \right\rfloor$.

Using the formula from Lemma 3.2 we can reduce the computation of $\begin{bmatrix} I \\ z \end{bmatrix}$ to smaller problems of the form $\begin{bmatrix} J \\ w \end{bmatrix}$ with $|J| \in \{1, 2\}$ and $w \in [1, z]$ in a divide-and-conquer manner; for these small problems we use the shortcuts of Lemma 3.1 running in constant time. By setting $m = n - 1$ in Lemma 3.2 we obtain the following formula:

$$\begin{bmatrix} (i_1, i_2, \ldots, i_n) \\ z \end{bmatrix} = \sum_{k=0}^{z} \begin{bmatrix} (i_1, i_2, \ldots, i_{n-1}) \\ k \end{bmatrix} \cdot \begin{bmatrix} (i_n) \\ z - k \end{bmatrix} \tag{1}$$

Since (i_n) is a single-valued sequence the factor $\begin{bmatrix} (i_n) \\ z - k \end{bmatrix}$ becomes either zero or one.

Algorithm 1: Solution Based on Divide and Conquer

Input: An array of upper bounds $I[1, n]$ and a target value $z \in \mathbb{Z}$

Output: $\begin{bmatrix} I \\ z \end{bmatrix}$ by calling `dnc(`$z, I[1, n]$`)`

```
1  Function dnc(z, I[l, r])
2      h ← Σ_{j=l}^{r} i_j          // 1 ≤ l ≤ r ≤ n represent the subinterval
         I[l, r] ⊆ I
3      if z = 0 ∨ z = h then return 1 // Lemma 3.1(b)
4      if z < 0 ∨ z > h then return 0 // Lemma 3.1(a)
5      if l = r then return 1 // Lemma 3.1(c)
6      if r = l + 1 then
7          m ← min(I[l], I[r])
8          if z > (I[l]+I[r])/2 then z ← h − z // Lemma 3.3
9          return min(z, m) + 1
10     v ← 0                                   // Invariant:  r > l
11     m ← (l+r)/2
12     for k ← 0 to z do
13         v ← v + dnc(k, I[l, m]) · dnc(z − k, I[m + 1, r])   // Lemma 3.2
14     return v
```

Alternatively, setting $m = n/2$ splits the bounds evenly in Lemma 3.2. We use this splitting in Algorithm 1. Algorithm 1 is derived from a preliminary version described by [7] based on work from [18]. Unfortunately, z has a crucial impact on the running time of this algorithm: If $T(n, z)$ gives the running time of `dnc(`$z, I[1, n]$`)` for an $I = (i_1, \ldots, i_n)$ and a $z < \sum_{j=1}^{n} i_j$, then $T(1, z) = 1$ and $T(n, z) = \sum_{k=1}^{z}(T(n/2, z - k) + T(n/2, k)) \approx 2zT(n/2, z) = (2z)^{\lg n}$, when using an average case estimation. Although it is easy to validate this algorithm, it is unsatisfying from a computational point of view. Its induced running time depends on the upper bounds I and the target value z (even if there may be heuristics that split up the sequence I advantageously). In contrast, we will develop an algorithm with a running time depending *only on the length* of the sequence of upper bounds, under the assumption that elementary arithmetic operations can be executed in constant time.

z	$\begin{bmatrix}(30)\\ z\end{bmatrix}$
$-\infty < z \leq -1$	0
$0 \leq z \leq 30$	1
$31 \leq z < \infty$	0

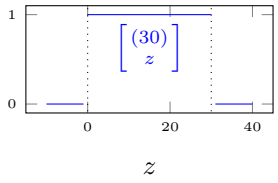

Figure 1: The piecewise-defined polynomial parameterized by z for the sequence $I = (30)$ obtained after the first round of Section 4.1. The table on the left gives the validity intervals of this piecewise-defined polynomial and its respective values. The figure on the right plots this piecewise-defined polynomial. A dotted vertical line demarcates the border of two validity intervals. The piecewise-defined polynomial continues with the value of zero at the left and the right borders of the plot.

z	$\begin{bmatrix}(30,50)\\ z\end{bmatrix}$
$-\infty < z \leq -1$	0
$0 \leq z \leq 30$	$z+1$
$31 \leq z \leq 49$	31
$50 \leq z \leq 80$	$81-z$
$81 \leq z \leq \infty$	0

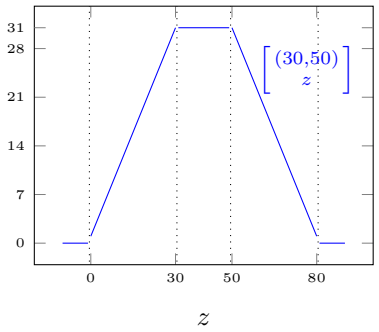

Figure 2: The piecewise-defined polynomial for the sequence $I = (30, 50)$ obtained after the second round of Section 4.1. The graphical presentation follows Figure 1.

4 The Algorithm

For a fixed sequence I, we show that $z \mapsto \begin{bmatrix}I\\ z\end{bmatrix}$ can be represented by a piecewise-defined polynomial in in z with rational coefficients of degree at most $|I| - 1$. More precisely, we compute a function $p : \mathbb{Z} \to \mathbb{Z}$ and a set of disjoint intervals $I_1, \ldots, I_m$ such that $\bigcup_{1 \leq k \leq m} I_k = \mathbb{Z}$, and $p|_{I_k}$ is a polynomial of degree at most $|I| - 1$ that coincides with $\begin{bmatrix}I\\ z\end{bmatrix}$ for every $z \in I_k$ and every $1 \leq k \leq m$. We call $I_1, \ldots, I_m$ the ***validity intervals*** of p.

We start with an idea derived from Lemma 3.1:

Lemma 4.1. For $I = (i_1)$, the piecewise-defined polynomial

$$P(z) =_{df} \begin{cases} 1 & \text{if } z \in [0, i_1] \\ 0 & \text{otherwise} \end{cases}$$

solves Problem 2.2. In the special case $i_1 < 0$ the polynomial is equal to zero.

z	$\begin{bmatrix}(30,50,10)\\ z\end{bmatrix}$
$-\infty < z \leq -1$	0
$0 \leq z \leq 9$	$\frac{z^2}{2} + \frac{3z}{2} + 1$
$10 \leq z \leq 30$	$11z - 44$
$31 \leq z \leq 40$	$-\frac{z^2}{2} + \frac{81z}{2} - 479$
$41 \leq z \leq 49$	341
$50 \leq z \leq 59$	$-\frac{z^2}{2} + \frac{99z}{2} - 884$
$60 \leq z \leq 80$	$-11z + 946$
$81 \leq z \leq 90$	$\frac{z^2}{2} - \frac{183z}{2} + 4186$
$91 \leq z < \infty$	0

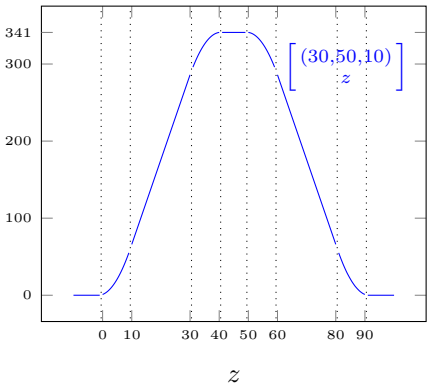

Figure 3: The piecewise-defined polynomial for the sequence $I = (30, 50, 10)$ obtained after the third round of Section 4.1. The graphical presentation follows Figure 1.

4.1 An Elaborated Example

Consider the sequence $I = (30, 50, 10)$. By applying Equation (1) we split up I recursively. This approach divides the algorithm up into three rounds — the number of rounds is equal to $|I|$. In each round, we construct a piecewise-defined polynomial function that represents the subsequence of I under consideration.

Round 1. We start with the sequence $I^1 = (30)$. By Lemma 4.1 we get

$$\begin{bmatrix}(30)\\ z\end{bmatrix} = \begin{cases} 0 & \text{for } z < 0, \\ 1 & \text{for } 0 \leq z \leq 30, \\ 0 & \text{for } z \geq 31. \end{cases}$$

We give a tabular and a graphical illustration in Figures 1 to 3 for each sequence I^j under consideration. Figure 1 depicts the function $\begin{bmatrix}(30)\\ z\end{bmatrix}$ by some dotted lines to emphasize that the function is discrete (mapping from $\mathbb{Z}$ to $\mathbb{N}_0$). The remaining parts of this figure depict the bounds of the next sequence:

Round 2. With an application of Lemma 3.3 we investigate

$$\begin{bmatrix}(30,50)\\ z\end{bmatrix} = \sum_{k=0}^{z} \begin{bmatrix}(30)\\ k\end{bmatrix} \cdot \begin{bmatrix}(50)\\ z-k\end{bmatrix}$$

by examining every summand $[0, z] \to \mathbb{Z}, k \mapsto \begin{bmatrix}(30)\\ k\end{bmatrix} \cdot \begin{bmatrix}(50)\\ z-k\end{bmatrix}$:

- For $z < 0$ the sum is empty and therefore evaluates to zero; we yield $\begin{bmatrix}(30,50)\\ z\end{bmatrix} = 0$ for $z < 0$.

- For $z \in [0, 30]$ the term $\begin{bmatrix}(50)\\ z-k\end{bmatrix}$ evaluates to one, and we have $\begin{bmatrix}(30,50)\\ z\end{bmatrix} = \sum_{k=0}^{z} \begin{bmatrix}(30)\\ k\end{bmatrix} = z + 1$.
- For $z \in [31, 49]$ the term $\begin{bmatrix}(30)\\ k\end{bmatrix}$ becomes zero for $k \in [31, 49]$, whereas $\begin{bmatrix}(50)\\ z-k\end{bmatrix}$ is always equal to one. Hence we have $\begin{bmatrix}(30,50)\\ z\end{bmatrix} = 31$.

The above cases are also illustrated in Figure 1: the intervals above of the function graph give an impression of the bounds of the summations under consideration.

The results for the remaining z-values follow by symmetry: with the help of Lemma 3.3 we conclude that $\begin{bmatrix}(30,50)\\ z\end{bmatrix}$ is symmetric with respect to $\frac{30+50}{2} = 40$. The next validity interval we have to consider is the interval $[0, 30]$ mirrored at 40, i.e. the interval $[50, 80]$:

- The reflection of the function $z \mapsto z + 1$ across the line $z \mapsto 40$ yields the function $z \mapsto 81 - z$,
- the mirror image of $]-\infty, -1]$ on 40 is $[81, \infty[$, and
- the mirror image of the constant function 0 is 0 again.

To sum up, we constructed a piecewise-defined, affine, linear function which determines the value of $\begin{bmatrix}(30,50)\\ z\end{bmatrix}$; it is visualized in Figure 2.

Round 3. Finally, we evaluate the sum

$$\sum_{k=0}^{z} \begin{bmatrix}(30,50)\\ k\end{bmatrix} \cdot \begin{bmatrix}(10)\\ z-k\end{bmatrix} = \begin{bmatrix}(30,50,10)\\ z\end{bmatrix}.$$

Since

$$\begin{bmatrix}(10)\\ z-k\end{bmatrix} = \begin{cases} 1 & \text{if } 0 \le z-k \le 10, \text{ and} \\ 0 & \text{otherwise,} \end{cases}$$

the previous sum can be written as $\sum_{k=z-10}^{z} \begin{bmatrix}(30,50)\\ k\end{bmatrix}$. Our task is to sum up the eleven consecutive values of $k \mapsto \begin{bmatrix}(30,50)\\ k\end{bmatrix}$ for $z - 10 \le k \le z$, which is easily computable by utilizing Gauß's summation formula for the first n natural numbers, i.e. $\sum_{i=0}^{n} = \frac{n(n+1)}{2}$. In order to apply this formula we have to examine the relative positions of the interval $[z-10, z]$ and the validity intervals of the polynomials of $\begin{bmatrix}(30,50)\\ z\end{bmatrix}$, i.e. the intervals $]-\infty, -1]$, $[0, 30]$, $[31, 49]$, $[50, 80]$ and $[81, \infty[$.

In the following, we exploit that the function $\begin{bmatrix}(30,50,10)\\ z\end{bmatrix}$ is symmetric with respect to $\frac{30+50+10}{2} = 45$ due to Lemma 3.3.

- The border cases are easy: We have $\left[\begin{matrix}(30,50,10)\\ z\end{matrix}\right] = 0$ for $z < 0$ and $\left[\begin{matrix}(30,50,10)\\ z\end{matrix}\right] = 0$ for $z > 90$.
- The next case ($z - 10 \in]-\infty, -1] \wedge z \in [0, 30]$) is the range $z \in [0, 9]$. Here we have

$$\sum_{k=z-10}^{z} \left[\begin{matrix}(30,50)\\ k\end{matrix}\right] = \sum_{k=0}^{z} \left[\begin{matrix}(30,50)\\ k\end{matrix}\right] = \sum_{k=0}^{z} (k+1) = \frac{z^2}{2} + \frac{3z}{2} + 1 =_{df} f(z).$$

  To obtain $\left[\begin{matrix}(30,50,10)\\ z\end{matrix}\right]$ for $z \in [81, 90]$ we reflect the function $f(z)$ across the (vertical) axis at 45; applying the reflection $\rho : z \mapsto 90 - z$ on $f(z)$ yields the function

$$f \circ \rho(z) = \frac{(90-z)^2}{2} + \frac{3(90-z)}{2} + 1 = \frac{z^2}{2} - \frac{183z}{2} + 4186.$$

- Now we consider the interval $[10, 30]$, where $[z - 10, z] \subseteq [0, 30]$. Here we get $\sum_{k=z-10}^{z} \left[\begin{matrix}(30,50)\\ k\end{matrix}\right] = \sum_{k=z-10}^{z} (k+1)$, and some simple arithmetic operations with Gauß' formula lead to the result $11z - 44$. By the symmetry argument we obtain $\left[\begin{matrix}(30,50,10)\\ z\end{matrix}\right] = 11(90 - z) - 44 = -11z + 946$ for $z \in [60, 80]$.
- Next we consider the case $z - 10 \in [0, 30] \wedge z \in [31, 40]$, i.e. $z \in [31, 40]$. For these values of z we get the equation $\sum_{k=z-10}^{z} \left[\begin{matrix}(30,50)\\ k\end{matrix}\right] = \sum_{k=z-10}^{30} (k + 1) + \sum_{k=31}^{z} 31$, which evaluates to $-\frac{z^2}{2} + \frac{81z}{2} - 479$. Again by Lemma 3.3 we have $\left[\begin{matrix}(30,50,10)\\ z\end{matrix}\right] = -\frac{(90-z)^2}{2} + \frac{81(90-z)}{2} - 479 = -\frac{z^2}{2} + \frac{99z}{2} - 884$ for $z \in [50, 59]$.
- If $z \in [41, 49]$ then $[z - 10, z] \subseteq [31, 49]$, hence $\left[\begin{matrix}(30,50,10)\\ z\end{matrix}\right] = \sum_{k=z-10}^{z} 31$, which gives the constant value 341.

Analogously to the previous step, the different positions of the sums' bounds are also sketched in Figure 2.

After the third round the computation of $\left[\begin{matrix}(30,50,10)\\ z\end{matrix}\right]$ finishes with the result shown in Figure 3. We stress again that we did *not* compute the actual values of the partition number (except the segments of the piecewise-defined polynomial with constant values). Instead, we constructed a piecewise-defined polynomial which solves Problem 2.2 by evaluating the piecewise-defined polynomial.

We divide the computation of the solution in two different units: A ***round*** is the computation of a new piecewise-defined polynomial. Each round is divided into steps. A ***step*** of the k-th round is the computation of a polynomial equal to the piecewise-defined polynomial of the k-th round on a certain domain.

4.2 Short Analysis of the Example

We now formalize the ideas introduced in the running example. In each round we applied the following property:

Lemma 4.2. Let $I = (i_1, i_2, \ldots, i_n)$ with $n \geq 2$ be a sequence of n non-negative integers and $z \in \mathbb{Z}$. Then

$$\begin{bmatrix} I \\ z \end{bmatrix} = \sum_{k=z-i_n}^{z} \begin{bmatrix} (i_1, i_2, \ldots, i_{n-1}) \\ k \end{bmatrix} \tag{2}$$

holds.

Proof. Let $J = (j_1, j_2, \ldots, j_n)$ be a partition of z with respect to I. Then clearly $\sum_{k=1}^{n-1} j_k = z - j_n$ holds. By fixing j_n there are exactly $\begin{bmatrix} (i_1, i_2, \ldots, i_{n-1}) \\ z - j_n \end{bmatrix}$ partitions. Since j_n has to be drawn from the interval $[0, i_n]$ we have the equality $\begin{bmatrix} I \\ z \end{bmatrix} = \sum_{k=0}^{i_n} \begin{bmatrix} (i_1, i_2, \ldots, i_{n-1}) \\ z - k \end{bmatrix}$, which leads to the claim after an elementary index shift. □ □

Algorithm 2: Abstract Outline of the Algorithm

Input: Array $I = (i_0, \ldots, i_{n-1}) \in \mathbb{N}_+^n$.
Output: A piecewise-defined polynomial that solves Problem 2.2

```
1 Function intpart(i_0, ..., i_{n-1})
2     p ← {λz ↦ 1 if z ∈ [0, i_0], z ↦ 0 otherwise}        // Lemma 4.1
3     for k ← 1 to n − 1 do
4         p ← { λz ↦ Σ_{j=z−i_k}^{z} p(j) }                // Apply Lemma 4.2
5     return p
```

Based on Lemma 4.2, Algorithm 2 represents the scaffold of the algorithm we will construct. The remaining part of this section shows how the summation of Equation (2) is done. For that we have to keep track of the validity intervals of the currently constructed piecewise-defined polynomial in each round. We need some rules that determine the validity intervals of the next round. In summary, we will study the following parts:

- Construction of the validity intervals (Section 4.3)
- Computation of the piecewise-defined polynomial in Equation (2) (Section 4.4)

4.3 Construction of the Validity Intervals

We start with some definitions:

Definition 4.3. An ***interval partition*** is a finite sequence $\mathbb{I} = (\mathcal{I}_0, \mathcal{I}_1, \ldots, \mathcal{I}_n)$ of nonempty intervals on $\mathbb{Z}$ such that

- $\max(\mathcal{I}_k) = \min(\mathcal{I}_{k+1}) - 1$ holds for all $k \in \{0, \ldots, n-1\}$ (this imposes the natural ordering on $\mathbb{I}$) and
- $\bigcup_{i=0}^{n} \mathcal{I}_i = \mathbb{Z}$.

Obviously, for an interval partition $(\mathcal{I}_0, \mathcal{I}_1, \ldots, \mathcal{I}_n)$ we have $\mathcal{I}_0 =]-\infty, z_1]$ and $\mathcal{I}_n = [z_2, \infty[$ for some integers z_1 and z_2. An interval partition $(\mathcal{I}_0, \mathcal{I}_1, \ldots, \mathcal{I}_n)$ is called ***normalized*** iff $\max(\mathcal{I}_0) = -1$ holds. The ***length*** of $\mathbb{I}$ is the number of intervals it contains, and is denoted by $|\mathbb{I}|$.

Example 4.4. Considering the example execution in Section 4.1, the validity intervals of our constructed polynomial in the second round form the interval partition $\mathbb{I}^e = (]-\infty, -1], [0, 30], [31, 50], [51, 80], [81, \infty[)$ with $|\mathbb{I}^e| = 5$. By Lemma 3.1, the constructed validity intervals always form a *normalized* interval partition if there exists at least one solution.

To get an idea on how we can formalize the splitting in each round, we introduce the following relation:

Definition 4.5. Let $\mathbb{I} = (\mathcal{I}_0, \mathcal{I}_1, \ldots, \mathcal{I}_n)$ be an interval partition, and $z \in \mathbb{N}_+$ a *positive* integer. We define the relation $\sim_z^{\mathbb{I}} \subseteq \mathbb{Z} \times \mathbb{Z}$ by $x \sim_z^{\mathbb{I}} y$ iff there exist some $i, j \in \mathbb{N}_0$ with $0 \leq i \leq j \leq n$ such that $x, y \in \mathcal{I}_j$ and $x - z, y - z \in \mathcal{I}_i$.

Lemma 4.6. The relation $\sim_z^{\mathbb{I}}$ of a (normalized) interval partition $\mathbb{I}$ and an arbitrary integer z is an equivalence relation whose equivalence classes form again a (normalized) interval partition. We call the resulting partition the ***interval partition induced by*** $\mathbb{I}$ ***and*** z, and denote it by $\mathbb{P}(\mathbb{I}, z)$.

Proof. Clearly, the relation $\sim_z^{\mathbb{I}}$ is reflexive and symmetric; it is transitive since the intervals of an interval partition are pairwise disjoint, i.e., $\sim_z^{\mathbb{I}}$ is an equivalence relation. A more powerful conclusion of the property of interval partitions is that $x \sim_z^{\mathbb{I}} y$ implies $x \sim_z^{\mathbb{I}} w$ for all w with $x \leq w \leq y$. Hence, the equivalence classes of $\sim_z^{\mathbb{I}}$ are intervals of $\mathbb{Z}$. These intervals are pairwise disjoint; otherwise $\sim_z^{\mathbb{I}}$ would not be transitive. Let us denote them by $\mathcal{J}_0, \ldots, \mathcal{J}_m$, sorted by their boundary values (natural ordering). In particular, $\mathcal{J}_0 = \mathcal{I}_0$ and $\mathcal{J}_m = [\min(\mathcal{I}_n) + z, \infty[$. For every $x + z \in \mathbb{Z}$ there is an i with $1 \leq i \leq n$ such that $x + z \in \mathcal{I}_i$. But then there is also a j with $1 \leq j \leq m$ such that $x \in \mathcal{J}_j$. Hence, the sequence $\mathbb{J} = (\mathcal{J}_0, \ldots, \mathcal{J}_m)$ is an interval partition, too. If $\mathbb{I}$ is *normalized*, then $\mathbb{J}$ is also normalized, since $\mathcal{J}_0 = \mathcal{I}_0$ holds. □ □

To sum up, $\mathbb{P}(\mathbb{I}, z)$ creates a finer partition of $\mathbb{Z}$. Every $\mathcal{J} \in \mathbb{P}(\mathbb{I}, z)$ is defined by the shape of exactly two intervals of $\mathcal{I}$. More precisely, there exists *exactly one* pair (i, j) with $1 \leq i \leq j \leq n$ such that every $x \in \mathcal{J}$ enjoys the properties $x - z \in \mathcal{I}_i$ and $x \in \mathcal{I}_j$. We call the *uniquely determined* intervals $\mathcal{I}_i$ and $\mathcal{I}_j$ the ***witness pair*** of $\mathcal{J}$ and denote them by $\psi_{(\mathcal{I},z)}(\mathcal{J})$; the set of witness pairs induces the injective mapping $\psi_{(\mathbb{I},z)} : \mathbb{P}(\mathbb{I}, z) \hookrightarrow \mathbb{I} \times \mathbb{I}$. If the context is clear, we abbreviate $\psi_{(\mathbb{I},z)}$ to ψ. In particular, the witness pair of $\mathcal{I}_0 = \mathcal{J}_0$ is given by the pair $(\mathcal{I}_0, \mathcal{I}_0)$; the witness pair of $\mathcal{J}_m = [\min(\mathcal{I}_n) + z, \infty[$ is $(\mathcal{I}_n, \mathcal{I}_n)$.

Our idea is to compute a piecewise-defined polynomial in each of the n rounds with Lemma 4.2. The validity intervals of the piecewise-defined polynomial in the $(k+1)$-th round are defined by $\mathbb{P}(\mathbb{I}^k, i_{k+1})$, where $\mathbb{I}^k$ are the validity intervals of the piecewise-defined polynomial of the k-th round.

Example 4.7. The interval partition induced by $\mathbb{I}^{\mathrm{e}}$ (defined in Example 4.4) and 10 is the sequence $(]-\infty,-1]$, $[0,10]$, $[11,30]$, $[31,40]$, $[41,50]$, $[51,60]$, $[61,80]$, $[81,90]$, $[91,\infty[)$. The first five members of this sequence were also considered in Section 4.1. For instance, we have $\psi_{(\mathcal{I}_e,10)}([0,10]) = (]-\infty,-1],[0,30])$, $\psi([11,30]) = ([0,30],[0,30])$ and $\psi([51,60]) = ([31,50],[51,80])$.

Regarding the maximum length of $\mathbb{P}(\mathbb{I},z)$, there are $\binom{n+1}{2} = n(n+1)/2$ possible witness pairs to choose. Fortunately, the following lemma gives a better upper bound:

Lemma 4.8. For an interval partition $\mathbb{I}$ and a positive integer z, we have the inequality $|\mathbb{P}(\mathbb{I},z)| \leq 2\,|\mathbb{I}| - 1$. We can compute $\mathbb{P}(\mathbb{I},z)$ in $\mathcal{O}(|\mathbb{I}|)$ time.

Proof. Let $(\mathcal{I}_0,\mathcal{I}_1,\ldots,\mathcal{I}_n) =_{df} \mathbb{I}$. We introduce a linear order on the set of witness pairs by $(\mathcal{I}_{i_1},\mathcal{I}_{j_1}) <_\psi (\mathcal{I}_{i_2},\mathcal{I}_{j_2}) \Leftrightarrow_{df} i_1 < i_2 \vee (i_1 = i_2 \wedge j_1 < j_2)$. The pair $(\mathcal{I}_0,\mathcal{I}_0)$ is the least element with respect to this order, and $(\mathcal{I}_n,\mathcal{I}_n)$ is its greatest element. By ordering the set of all interval witnesses according to $<_\psi$ we obtain a chain $(\mathcal{I}_{i_0},\mathcal{I}_{j_0}) <_\psi (\mathcal{I}_{i_1},\mathcal{I}_{j_1}) <_\psi \ldots <_\psi (\mathcal{I}_{i_m},\mathcal{I}_{j_m})$ with $i_0 = j_0 = 0$ and $i_m = j_m = n$. For two consecutive pairs $(\mathcal{I}_{i_k},\mathcal{I}_{j_k})$ and $(\mathcal{I}_{i_{k+1}},\mathcal{I}_{j_{k+1}})$ of this sequence we have $(i_{k+1}+j_{k+1}) - (i_k+j_k) \in \{1,2\}$ due to the underlying definition of $\sim_z^{\mathbb{I}}$. Since $i_0 + j_0 = 0$ and $i_m + j_m = 2n$ the claim follows.

The computation of $\mathbb{P}(\mathbb{I},z)$ can be done by merging an interval $\mathbb{J}$ of length z from the left to the right (like with a sweep line) into $\mathbb{I}$ while tracking the events when the bounds of $\mathbb{J}$ enter or leave an interval in $\mathbb{I}$. (A more detailed and practical optimized algorithm is described in Section 7.) □ □

After these definitions and lemmas concerning the partition of $\mathbb{Z}$ into intervals we now turn our attention back to the construction of the piecewise-defined polynomial.

4.4 Construction of the Piecewise-Defined Polynomial

Our main idea is based on a classical, well-known result named after Faulhaber:

Lemma 4.9 ([6, 13])**.** Given $p,n \in \mathbb{N}_+$, ***Faulhaber's formula*** states that

$$\sum_{i=1}^{n} i^p = \sum_{j=0}^{p} \binom{p}{j} \frac{B_{p-j}}{j+1} n^{j+1}, \tag{3}$$

where B_l denotes the l-th ***Bernoulli number***. Although $\frac{B_{p-j}}{j+1} \in \mathbb{Q}$ with $j \in \mathbb{N}_0$ is a fractional term, the polynomial on the right hand side resolves to an integer for every $n \in \mathbb{N}_+$.

For Lemma 4.9 to be of use, we need to compute the Bernoulli numbers.

Lemma 4.10. The first n Bernoulli numbers can be computed in $\mathcal{O}(n^2)$ time. The binomial coefficients $\binom{j}{k}$ for $k,j \in \mathcal{O}(n)$ can be computed in the same time.

Proof. The Bernoulli numbers are given by the recurrence relation $B_0 = 1$ and $B_n = -\frac{1}{n+1}\sum_{k=0}^{n-1}\binom{n+1}{k}B_k$. In the formula for B_n, the values for each B_k can be

looked up in constant time using the previously computed numbers. Also, the values of $\binom{n+1}{k}$ can be computed in constant time per summand by $\binom{n+1}{0} = 1$ and $\binom{n+1}{k+1} = \frac{n+1-k}{k+1}\binom{n+1}{k}$. Since the sum for B_i consists of i summands, the overall running time is in $\mathcal{O}(n^2)$. □ □

In the further course, the Bernoulli numbers are computed in a precomputation step whose running time vanishes compared to other computations.

A direct consequence of Equation (3) is the following lemma:

Lemma 4.11. Given two arbitrary numbers $n, \gamma \in \mathbb{N}_0$, the function $\sigma_\gamma(z) : [\gamma, \infty[\to \mathbb{Z}$, defined by $\sigma_\gamma(z) = \sum_{k=0}^{z-\gamma} k^n$ is a polynomial in z with rational coefficients of degree $n+1$. Given the first n Bernoulli numbers and the values $\binom{j}{k}$ for $1 \leq k \leq j \leq n+1$, the coefficients of σ_γ can be computed in $\mathcal{O}(n^2)$ time.

Proof. By Faulhaber's formula, there are some coefficients $a_0, a_1, \ldots, a_{n+1} \in \mathbb{Q}$ with $\sum_{k=0}^{z-\gamma} k^n = \sum_{l=0}^{n+1} a_l \cdot (z-\gamma)^l$. Due to the binomial theorem, this sum is equal to $\sum_{l=0}^{n+1} \left(a_l \cdot \sum_{m=0}^{l} \binom{l}{m} \cdot z^m \cdot (-\gamma)^{l-m} \right)$. We define $\beta_{lm} =_{df} \binom{l}{m} \cdot (-\gamma)^{l-m} \in \mathbb{Z}$ and obtain the term $\sum_{l=0}^{n+1} \left(a_l \cdot \sum_{m=0}^{l} \beta_{lm} \cdot z^m \right)$. By setting $\delta_m =_{df} \sum_{l=m}^{n+1} a_l \cdot \beta_{lm} \in \mathbb{Q}$, the equation gets simplified to $\sum_{m=0}^{n+1} \delta_m \cdot z^m$. The running time is due to the computation of δ_m for every $0 \leq m \leq n+1$. □ □

The following corollary is a direct conclusion of the previous lemma involving some simple arithmetic operations:

Corollary 4.12. Let p be a polynomial in $\mathbb{Q}$ of degree n. For a natural number γ, we have:

a) The function $\sigma_{\geq\gamma} : [\gamma, \infty[\to \mathbb{R}$ with $\sigma_{\geq\gamma}(z) = \sum_{i=\gamma}^{z} p(i)$ is a polynomial of degree at most $n+1$.

b) The function $\sigma_{\leq\gamma} : [0, \gamma] \to \mathbb{R}$ defined by $\sigma_{\leq\gamma}(z) = \sum_{i=z}^{\gamma} p(i)$ is a polynomial of degree at most $n+1$.

c) The function $\sigma_{-\gamma} : [\gamma, \infty[\to \mathbb{R}$ with $\sigma_{-\gamma}(z) = \sum_{i=z-\gamma}^{z} p(i)$ is a polynomial in $\mathbb{Q}$ of degree at most n.

Given the first n Bernoulli numbers and the values $\binom{j}{k}$ for $1 \leq k \leq j \leq n+1$, the polynomials $\sigma_{\geq\gamma}, \sigma_{\leq\gamma}$, and $\sigma_{-\gamma}$ can be constructed in $\mathcal{O}(n^2)$ time.

Proof. Assume that $p(x) = x^n$, and let σ_γ be defined as in Lemma 4.11. The coefficients of $\sigma_{\geq\gamma} = \sigma_0(z) - \sigma_0(\gamma - 1)$, $\sigma_{\leq\gamma} = \sigma_0(\gamma) - \sigma_0(z-1)$, and $\sigma_{-\gamma} = \sigma_0(z) - \sigma_{\gamma-1}(z)$ can be easily determined using the coefficients from the proof of Lemma 4.11. In particular, $\delta_{n+1} = a_n$ (a_n and δ_{n+1} as defined in the proof of Lemma 4.11), since $\beta_{n+1,n+1} = 1$. Hence, the highest coefficients of $\sigma_0(z)$ and $\sigma_{\gamma-1}(z)$ are the same.

The claim follows for a general polynomial p by basic arithmetic operations. □ □

Now we are ready to prove the main result of this section:

Theorem 4.13. Let $I = (i_1, i_2, \ldots, i_n)$ be a sequence of n positive integers. Then there exists a piecewise-defined polynomial $p : \bigcup_{j=1}^{r} \mathcal{I}_r \to \mathbb{N}_0$ such that

(a) $r \leq 2^n + 1$,

(b) its validity intervals $(\mathcal{I}_0, \mathcal{I}_1, \ldots, \mathcal{I}_r)$ form a normalized interval partition,

(c) for each $1 \leq j \leq r$, $p_j =_{df} p|_{I_j}$ is a polynomial of degree at most $n-1$,

(d) p solves Problem 2.2, i.e. $\begin{bmatrix} I \\ z \end{bmatrix} = p(z)$ for every $z \in \mathbb{Z}$.

Proof. The proof is done via induction over $|I|$.

Induction base. This is exactly described in Lemma 4.1.

Induction step. Consider a sequence $I^{n+1} = (i_1, i_2, \ldots, i_{n+1})$ of $n+1$ positive integers. By the induction hypothesis, let p_n be the piecewise-defined polynomial of $I^n =_{df} (i_1, i_2, \ldots, i_n)$ such that it holds the premises described in the claim; we denote its validity intervals by $\mathbb{I}^n = (\mathcal{I}_0^n, \mathcal{I}_1^n, \ldots, \mathcal{I}_r^n)$ and the partition induced by $\mathbb{I}^n$ and i_{n+1} with $(\mathcal{I}_0^{n+1}, \mathcal{I}_1^{n+1}, \ldots, \mathcal{I}_w^{n+1}) =_{df} \mathbb{P}(\mathbb{I}^n, i_{n+1})$. In the following, we construct a piecewise-defined polynomial solving Problem 2.2 for I^{n+1}. This piecewise-defined polynomial is constructed from p_n and i_{n+1}. It consists of at most $w \leq 2\,|\mathbb{I}^n| - 1 \leq 2^{n+1} + 1$ validity intervals due to Lemma 4.8 and the induction hypothesis. It is left to show the construction that solves (c) and (d). We partition the sum of Equation (2), applied on p_n, into the validity intervals of p_n such that we have sums over sums of polynomials of degree at most $n-1$: Given an m with $1 \leq m \leq w$, we consider the interval $[s,t] =_{df} \mathcal{I}_m^{n+1} \in \mathbb{P}(\mathbb{I}^n, i_{n+1})$, and denote its witness pair of intervals from $\mathbb{I}^n$ by $([s_\ell, t_\ell], [s_h, t_h]) =_{df} (L, H) =_{df} \psi([s,t])$, where $L = \mathcal{I}_\ell^n$ and $H = \mathcal{I}_h^n$ for some $1 \leq \ell, h \leq r$. Fix an arbitrary $z \in [s,t]$. By the definition of z and L we have $z - i_{n+1} \in L$. Equation (2) leads to $\begin{bmatrix} I^{n+1} \\ z \end{bmatrix} = \sum_{k=z-i_{n+1}}^{z} \begin{bmatrix} I^n \\ k \end{bmatrix}$, where $\begin{bmatrix} I^n \\ k \end{bmatrix} = p_n(k)$ is the piecewise-defined polynomial of the n-th round (i.e., the preceeding round). To compute the above equation, we have to consider two cases:

- Let $L \neq H$ hold. Then we have $s_\ell < t_\ell < s_h < t_h$, and we can split up the above sum into

$$\sum_{k=z-i_{n+1}}^{z} \begin{bmatrix} I^n \\ k \end{bmatrix} = \sum_{k=z-i_{n+1}}^{t_\ell} \begin{bmatrix} I^n \\ k \end{bmatrix} + \sum_{k \in \mathcal{I}_j^n; \ell < j < h} \begin{bmatrix} I^n \\ k \end{bmatrix} + \sum_{k=s_h}^{z} \begin{bmatrix} I^n \\ k \end{bmatrix}.$$

  The second sum of the right hand side is a constant, and the first and the third sum can be written as polynomials in z of degree at most n according to the induction hypothesis and Corollary 4.12.

Algorithm 3: Constructing a piecewise-defined polynomial

Input: Array $I = (i_0, \ldots, i_{n-1}) \in \mathbb{N}_+^n$.
Output: A piecewise-defined polynomial that solves Problem 2.2

Function `intpart`$(i_0, \ldots, i_{n-1})$
  $\mathbb{I}^0 \leftarrow ([0, i_0])$
  $p \leftarrow \{\lambda.z \mapsto 1 \text{ if } z \in [0, i_0], z \mapsto 0 \text{ otherwise}\}$ `// Lemma 4.1`
  for $k \leftarrow 1$ *to* $n-1$ **do**
    $S \leftarrow \mathbb{P}(\mathbb{I}^{k-1}, i_k)$ `// Partition induced by` $\mathbb{I}^{k-1}$ `and` i_k
    $p' \leftarrow$ associative array (Interval $\to$ Polynomial) `// create a new piecewise-defined polynomial`
    $\mathbb{I}^k \leftarrow \emptyset$
    foreach $(L, H) \in \psi_{(\mathbb{I}^{k-1}, i_k)}(S)$ **do** `// Fetch witness pairs in natural order`
      Let $[s_\ell, t_\ell] =_{df} L$, $[s_h, t_h] =_{df} H$, $E =_{df} \psi^{-1}_{(\mathbb{I}_{k-1}, i_k)}(L, H)$, $\left(\mathcal{I}_0, \ldots, \mathcal{I}_{|\mathbb{I}^{k-1}|}\right) =_{df} \mathbb{I}^{k-1}$
      Hence $\exists \ell, h$ with $0 \le \ell \le h \le \left|\mathbb{I}^{k-1}\right|$ with $L = \mathcal{I}_\ell \wedge H = \mathcal{I}_h$
      $\mathbb{I}^k \leftarrow \mathbb{I}^k \cup \{E\}$
      Let $I_{k-1} =_{df} (i_0, \ldots, i_{k-1})$
      if $L = H$ **then** `// Apply Lemma 4.2 and Corollary 4.12`
        Add $\left(E, z \mapsto \sum_{j=z-i_k}^{z} p(j)\right)$ to p'
      else `// Split sum of Lemma 4.2 up in three parts`
        $U(z) \leftarrow \sum_{j=s_h}^{z} \begin{bmatrix} I_{k-1} \\ j \end{bmatrix}$
        $C \leftarrow \sum_{j \in \mathcal{I}_m : \ell < m < h} \begin{bmatrix} I_{k-1} \\ j \end{bmatrix}$ `// independent of` z
        $D(z) \leftarrow \sum_{j=z-i_k}^{t_\ell} \begin{bmatrix} I_{k-1} \\ j \end{bmatrix}$
        Add $(E, z \mapsto U(z) + C + D(z))$ to p'
    $p \leftarrow p'$
  return p

- In the case $L = H$, the induction hypothesis leads to $\sum_{k=z-i_{n+1}}^{z} \begin{bmatrix} I^n \\ k \end{bmatrix} = \sum_{k=z-i_{n+1}}^{z} p(k)$ for some polynomial p with degree at most $n-1$. This sum is a polynomial in z of degree at most $n-1$, due to Corollary 4.12(c).

We have shown (c); (d) follows by construction. □ □

Algorithm 3 gives an implementation of how the summation in Algorithm 2 can be realized by following the proof of Theorem 4.13: Given the piecewise-defined polynomial of the $(k-1)$-th round, we perform the following steps in the k-th round:

I create the interval partition $\mathbb{I}$ induced by its validity intervals and the upper bound of the k-th round,

II compute the *new* validity intervals by the witness pairs of $\mathbb{I}$,

III split up the sum of Equation (2) for each *new* validity interval like in proof of Theorem 4.13,

IV create a new piecewise-defined polynomial by the new validity intervals and the summations, and finally

V continue with the $(k+1)$-th round.

5 Running Time

For the running time analysis we assume that the basic arithmetic operations addition, subtraction, multiplication and division can be carried out in constant time.

We show in this section the following theorem:

Theorem 5.1. Given a fixed sequence $I = (i_1, i_2, \ldots, i_n)$ of n integer numbers there is a data structure computing the value $\begin{bmatrix} I \\ z \end{bmatrix}$ for every $z \in \mathbb{Z}$ in $\mathcal{O}(n)$ time, independent of z. The data structure can be constructed in $\mathcal{O}\big(2^n n^3\big)$ time.

We use the considerations of the previous section to develop a data structure that evaluates $\mathbb{Z} \to \mathbb{N}_0, z \mapsto \begin{bmatrix} I \\ z \end{bmatrix}$ for a fixed sequence $I = (i_1, i_2, \ldots, i_n)$ with $n \in \mathbb{N}_+$.

5.1 Theoretical Analysis

We focus on three phases:

(a) The construction of helping data structures like the binomial coefficients and the Bernoulli numbers (see Lemma 4.10) in the pre-computation phase,

(b) the actual computation of the piecewise-defined polynomials, and

(c) the evaluation for a given $z \in \mathbb{Z}$.

(a) Precomputation. We compute the coefficients of Faulhaber's formula (given in Lemma 4.9) for all exponents between 1 and $n-1$, and store them in a lookup table. Since these coefficients are based on the Bernoulli numbers and the binomial coefficients, we store all binomial coefficients $\binom{m}{l}$ with $0 \leq l \leq m \leq n+1$ and all Bernoulli numbers up to n in lookup tables, too. A naïve approach takes polynomial time in n, which will not influence the asymptotic running time. Alternatively, the computation of the coefficients can be postponed until they are actually needed. Such a lazy behaviour speeds up the initialisation at the price of a later, practically slower evaluation.

(b) Computation of the piecewise-defined polynomial. We represent a piecewise-defined polynomial by a list of pairs; each pair consists of a validity interval and its respective polynomial. The list is sorted by the beginning positions of the validity intervals. Having the list of pairs in a sorted order, a piecewise-defined polynomial with a list of size m can be evaluated in $\mathcal{O}(m)$ time by means of a binary search on the validity interval bounds.

Speaking of the piecewise-defined polynomials constructed in Algorithm 3, we do not have to sort the list of any constructed piecewise-defined polynomial; we fill the lists in ascending order by processing the witness pairs as highlighted in the algorithm above. The constructed piecewise-polynomial of the k-th round has at most $2^k + 1$ validity intervals, thus we can find the validity interval containing a given value in $\mathcal{O}\big(\lg(2^k + 1)\big) = \mathcal{O}(k)$ time.

The most demanding part of Algorithm 3 is the computation of the coefficients of the polynomial defined on an interval $\mathcal{I}_m^k$ of $\mathbb{I}^k = \mathbb{P}(\mathbb{I}^{k-1}, i_k)$, where $1 \leq k \leq n$ and $1 \leq m \leq \big|\mathbb{I}^k\big|$. We make again a difference whether L and H coincide:

- If $L \neq H$ the algorithm computes the three sums $U(z), C$ and $D(z)$. The computation of the coefficients of the first and last sum can be carried out along the lines of the proof of Lemma 4.11 and Corollary 4.12 in $\mathcal{O}\big(k^3\big)$ time: for a monomial of the form $c_j z^j$ with $c_j \in \mathbb{Q}$ and $j \in \mathbb{N}_0$ the coefficients can be computed in $\mathcal{O}\big(j^2\big)$ time, and we do so for all monomials $c_0, c_1 z, \ldots, c_{k-1} z^{k-1}$ of order at most $k-1$, due to Theorem 4.13c).

  Regarding the constant sum in the middle, we do not compute C in every round from scratch, but handle this sum by a stepping technique: we process the validity intervals in the ascending order described in the proof of Lemma 4.8. Between two consecutive steps (stepping from $\mathcal{I}_m^k$ to $\mathcal{I}_{m+1}^k$) we look at how C changes: either C keeps the same value, or it is incremented/decremented by a value of $\sum_{j \in \mathcal{I}_l^{k-1}} \begin{bmatrix} (i_1, i_2, \ldots, i_{k-1}) \\ j \end{bmatrix}$ for *exactly* one interval $\mathcal{I}_l^{k-1}$ of $\mathbb{I}^{k-1}$. The change can be computed analogously as sketched for $U(z)$ and $D(z)$ in $\mathcal{O}\big(k^3\big)$ time.

- In the case $L = H$ we can compute the coefficients analogously to above in $\mathcal{O}\big(k^3\big)$ time, too.

According to Theorem 4.13a) we have at most $2^k - 1$ validity intervals. Therefore, the running time for the k-th iteration is $\mathcal{O}\big(2^k k^3\big)$. Since we perform n iterations, the total running time is $\mathcal{O}\Big(\sum_{k=1}^{n} 2^k k^3\Big) = \mathcal{O}\big(2^n n^3\big)$.

(c) Evaluation time. The validity interval containing a given target value z can be found by binary searching in $\mathcal{O}(\lg(2^n + 1)) = \mathcal{O}(n)$ time. The piecewise-defined polynomial on the respective validity interval is a polynomial of degree $n-1$. This polynomial can be evaluated in $\mathcal{O}(n-1)$ time using Horner's method. Altogether, the computation of $\begin{bmatrix} I \\ z \end{bmatrix}$ is carried out in time linear to n. This concludes the proof of Theorem 5.1.

6 Parallel Execution

It is easy to parallelize the algorithm using a pre-computation step:

Theorem 6.1. Given $\pi = \mathcal{O}(n^3)$ processors, we can construct the data structure of Theorem 5.1 in parallel. If all processors are homogeneous, the computation time is $\mathcal{O}\left(2^n \frac{n^3}{\pi}\right)$ with high probability.

Our idea is to split up the computation of Algorithm 3 into a lightweight sequential pre-computation phase and a heavyweight parallel computation phase.

Sequential Pre-Computation. All validity intervals and witness pairs can be computed without having the polynomials at hand. Hence it is eligible to pre-compute the validity intervals of the k-th piecewise-defined polynomial for *every* $1 \leq k \leq n$, before starting with the construction of the polynomials (in particular, their coefficients). The pre-computation computes the validity intervals of each piecewise-defined polynomial in $\mathcal{O}(2^n)$ time. It also constructs the n piecewise-defined polynomials, but omits to compute the coefficients of its polynomials. We call these half-baked piecewise-defined polynomials ***skeletons***. The task of the subsequent parallel computation is to fill these skeletons. To transmit these tasks to the parallel phase we describe them by the constructs of ***futures*** and promises [2]: Each skeleton of a piecewise-defined polynomial p is stored as a list of pairs; each pair stores a validity interval with its respective future. Each future describes a polynomial that has to be computed in the parallel phase. This description contains the respective validity interval and the witness pair. The future can depend on other futures, i.e., on the computation of a polynomial belonging to a piecewise-defined polynomial of a former round. The pre-computation phase ends when the futures of all skeletons are described as tasks put into a queue.

Parallel Computation. In the pre-computation phase, the main thread pushes tasks into a queue. After the main thread has finished this phase, the other threads start processing those pushed tasks. On removing a task from the queue, a thread processes the task instantaneously. This fails if the future of this task depends on a not-yet computed polynomial. This polynomial has to be currently processed by another thread due to the ordering of the tasks put into the queue. The tasks are ordered in such a way that the k-th skeleton is transformed into the k-th piecewise-defined polynomials ($1 \leq k \leq n$), in ascending order of k. Every task of the k-th skeleton ($1 \leq k \leq n$) takes $\mathcal{O}(2^k k^3)$ time, assuming that the computation is wait-free.

Figure 4 shows both phases with the blockings symbolized by lock symbols. The blockings are present in a few places making the execution non-wait free:

- Taking a task from the queue is locked by a mutex. Alternatively, we could create a queue for each thread using a load-balancing (like [17]) or a work stealing technique.
- The memoization data structure mentioned above is equipped with a read-write lock in case that the associative array is sensitive for adding elements simultaneously. Alternatively, we could endow each thread with its own associative array, resulting in a higher work load.

- The futures of the k-th skeleton can only be computed if its dependencies on the futures of the $(k-1)$-th polynomial are fulfilled.

The parallel execution depends on the resolution of the dependencies of the futures. Informally speaking, the chance of processing a future without waiting correlates with the number of validity intervals. Given that the piecewise-defined polynomial p_k of the k-th round has much more validity intervals than there are threads, it is likely that most parts of p_k have already been computed when a threads starts to compute a future of p_{k+1}. Since we compute the polynomials of p_k ascendingly sorted by their respective validity intervals, it is likely that p_{k+1} can be processed wait-free.

Let $\pi \in \mathcal{O}(n^3)$ be the number of threads. To analyze the running time, we assume that p_n of the last round has $\Theta(2^n)$ validity intervals. Otherwise, if there are $m = \mathrm{o}(2^n)$ validity intervals, then the construction algorithm of Theorem 5.1 runs in $\mathcal{O}(mn^3)$ time, and $\mathcal{O}(mn^3) \subsetneq \mathcal{O}\left(2^n \frac{n^3}{\pi^2}\right)$, i.e., no parallel execution is necessary to get the postulated bounds.

Given that p_n has $\Theta(2^n)$ many validity intervals, the computation of a future is likely wait-free after the $\mathcal{O}(\lg \pi)$-th round if we assume that the processing power of all threads is homogeneous. In this case, the wait-free part is dominating since the overall time is

$$\mathcal{O}\left(\underbrace{2^n}_{\text{precomp}} + \underbrace{\pi(\lg \pi)^3}_{\text{wait}} + \underbrace{\sum_{k=\pi+1}^{n} 2^k \frac{k^3}{\pi}}_{\text{wait-free}}\right) = \mathcal{O}\left(2^n \frac{n^3}{\pi}\right)$$

with high probability.

7 Practical Improvements of the Construction

First of all, we intercept special cases of upper bounds with the following properties:

- $\begin{bmatrix} (i_{\phi(1)}, \ldots, i_{\phi(n)}) \\ z \end{bmatrix} = \begin{bmatrix} (i_1, \ldots, i_n) \\ z \end{bmatrix}$, where $\phi : [1, n] \to [1, n]$ is a permutation,
- $\begin{bmatrix} (0, i_2, \ldots, i_n) \\ z \end{bmatrix} = \begin{bmatrix} (i_2, \ldots, i_n) \\ z \end{bmatrix}$
- $\begin{bmatrix} (i_1, i_2, \ldots, i_n) \\ z \end{bmatrix} = 0$ if $i_1 < 0$
- $\begin{bmatrix} I \\ z \end{bmatrix} = \binom{n}{z}$ if $(i_1, i_2, \ldots, i_n) = (1, \ldots, 1)$.

We consider the following special cases:

- If there is a negative upper bound, we let the algorithm terminate prematurely, returning the polynomial $p = 0$.

- We skip every bound that is equal to zero.
- We count the number d of bounds with the value one, and omit those bounds during the construction of the piecewise-defined polynomial p built on the $n-d$ remaining upper bounds. With Lemma 3.2 we can compute $\begin{bmatrix} I \\ z \end{bmatrix} = \sum_{k=0}^{\min(z,d)} \binom{d}{k} p(z-k)$ in $\mathcal{O}((n-d)\min(z,d))$ time. We pay a little more evaluation time ($\mathcal{O}(n^2)$ time) for speeding up the construction time considerably.

From now on, it is left to deal with sequences of type $I = (i_1, \ldots, i_n)$ with $i_j > 1$ for all $1 \leq j \leq n$.

Compact Representation of Normalized Interval Partitions. To slim down the space usage we store just the upper boundaries $(z_1, \ldots, z_j)$ of every normalized interval partition $\mathbb{I}^k = (]-\infty, -1], [0, z_1], [z_1+1, z_2], \ldots, [z_{j-1}+1, z_j], [z_j+1, \infty[)$. We can use this interval partition representation for the validity intervals, too: Let us assume that we have finished the $(k-1)$-th round. In the k-th round, we compute the interval partition $\mathbb{I}^k$ induced by $\mathbb{I}^{k-1}$ and i_k by means of a sweep line algorithm in $\mathcal{O}(|\mathbb{I}^{k-1}|)$ time (remember Lemma 4.8).

In more detail, let $(z_1, \ldots, z_j) =_{df} \mathbb{I}^{k-1}$. We compute the representation of $\mathbb{I}^k$ by merging the *ordered* sequence $(i_k - 1, z_1 + i_k, z_2 + i_k, \ldots, z_j + i_k)$ with the *ordered* sequence $(z_1, z_2, \ldots, z_j)$, while removing duplicates and any zeros. The merging gives an *ordered* sequence of integers representing $\mathbb{I}^k$ in $\mathcal{O}(j)$ time. During this merging process we determine the witness pair $L, H \in \mathbb{I}^{k-1}$ for each interval $\mathcal{I}_m^k$ of $\mathbb{P}(\mathbb{I}^{k-1}, i_{k+1})$ with $1 \leq m \leq |\mathbb{I}^k|$ in constant time by a simple case distinction.

Symmetric Property. We take advantage of the idea in Lemma 3.3: it suffices to compute a piecewise-defined polynomial that answers Problem 2.2 for $0 \leq z \leq \kappa =_{df} \left\lceil \sum_{j=1}^{n} \frac{i_j}{2} \right\rceil$. Hence, we can omit computing the polynomials on the domains $[\kappa+1, 2\kappa]$. Given that Algorithm 3 loops over an increasingly sorted list of witness pairs, due to the above considerations, we `break` the `foreach` loop whenever the lower witness L does not intersect with $[0, \kappa]$. This shortcut does not deteriorate the solution: neither the new coefficients nor the new validity intervals depend on the values of the old piecewise-defined polynomial outside the witness pair of the respective, new validity interval, at each round of the algorithm. Finally, for answering a query with $\kappa < z \leq 2\kappa$ we evaluate the piecewise-defined polynomial at $2\kappa - z$. To sum up, exploiting the symmetric property reduces the running time of the algorithm to at least half of it.

Memoization technique. The computation of $p \mapsto \sigma_0(p)$ for a polynomial p in $\mathbb{Q}$ is run $\mathcal{O}(2^n n)$ times. In practice, we often evaluate this function for the same polynomial; a memoization of this function comes in handy. We deploy a memoization data structure retrieving $\sigma_0(p)$ for a given polynomial p (it either computes and stores $\sigma_0(p)$, or looks up $\sigma_0(p)$ if it has already been computed).

8 Experimental Results

We compared our Faulhaber based data structure of Algorithm 3 with the naïve algorithm of Algorithm 1. The implementations of both algorithms are written in C++14, and are available at `https://github.com/koeppl/integer_partition`.

Elementary Data Types. We focused on the exact calculation of the solution. The algorithm based on Faulhaber's formula needs to operate with rational numbers. On the one hand, we can express a rational number by a floating point data type approximately, running into numerical inaccuracies at high dimensions or huge upper bound values. On the other hand, we can express the numerator and the denominator with integer types. Although the latter approach does not have to deal with rounding errors, it is slower on common computers. We went a step further and employed integer types of arbitrary bit lengths to remedy arithmetic overflows. For that purpose we used the data types of the GNU Multiple Precision Arithmetic Library (see `https://gmplib.org`). It goes without saying that this approach is slower than working with elementary numeric data types suitable for small instances.

Precomputation. We precomputed the first 200 binomial coefficients in a lookup table, such that we can answer $\binom{n}{k}$ for $n \leq 200, k \leq n$ in constant time.

Memoization. We realized the memoization technique by deploying an associative array for each polynomial degree. The associative arrays are implemented with red-black trees. A better candidate would have been a geometric data structure like a kd-tree. Neither a hash table nor a standard kd-tree did (considerably) improve the running time of our experiments.

Rarely used polynomials tend to clutter the dictionary. On some instances, a good approach would be to track the usage of the stored polynomials such that the associative array can remove old unused entries on a regular basis.

Setup. The experiments were conducted on a machine with 32 GB of RAM, an Intel Xeon CPU `E3-1271 v3` and a Samsung SSD `850 EVO 250GB`. The operating system was a 64-bit version of Ubuntu Linux 14.04 with the kernel version 3.13. We used up to four cores for the execution. The source code was compiled using the GNU compiler `g++ 6.2.0` with the compile flags `-O3 -march=native -DNDEBUG`. Under these circumstances we obtained the results from Figure 5. In the following, we call the Faulhaber based data structure Faulhaber, and the naïve algorithm Naïve. The running times of Faulhaber include the pre-computation of the aforementioned coefficients, and the construction of the piecewise-defined polynomials. The sequential variant of Faulhaber is not considerably faster as the parallel version of Faulhaber with one thread, i.e., we do not distinguish between the sequential version and the parallel version running with one thread in the experiments. We aborted a test run (speaking of Naïve) after one hour.

In order to test and demonstrate some properties we constructed the examples No. 1–17,23–32 by hand and used randomly generated instances to test the average behaviour (No. 18 and No. 19).

	Number of Threads			
No.	1	2	3	4
14	763	404	284	249
15	2978	1535	1062	950
16	15622	7949	6149	4714
17	15618	7927	5456	4335
22	182	124	83	74
26	119	70	54	45
27	523	277	195	154
28	2384	1220	834	790
29	10611	5323	3658	2826
30	46993	23633	16355	12609

Table 1: Running time of Faulhaber with different numbers of threads. Time is in milliseconds.

The experiments reveal expected and interesting results:

- On small instances Naïve achieves better running times because it does not need to construct a data structure with an expensive construction time (see No. 1, 2, and 3).
- Naïve runs fast on instances with a small target value, even if the upper bounds are great (see No. 5,6, 9, 11, and 16). In this case the sum from Lemma 3.2 has only a small number of summands.
- The running time of Faulhaber correlates with the number of the upper bounds (cf. No. 1-4, 5-10, and No. 11/12; the last two instances show that the running time tends to slow down on larger numbers).
- Faulhaber has a running time depending also on the number of validity intervals in the constructed data structure. For a sequence $(i_1, i_2, \ldots, i_n)$ with $i_1 = i_2 = \ldots = i_n$ there are exactly $2n+1$ validity intervals in the final result (see No. 5–7,20–22). The sequences from No. 9-12 are constructed such that they have the maximal number of validity intervals in the final data structure. This explains the difference between No. 13 and No. 14. A drastic example for this phenomenon is given by No. 20 and No. 27.
- The examples No. 23–30 (which have the maximal number of validity intervals in the final data structure) show the hyperexponential growth of the runtime (recall the data structure construction time of $\mathcal{O}\big(2^n n^3\big)$ in Theorem 5.1) of Faulhaber. However, it performs for big input instances and great target values better than Naïve.
- Seemingly surprisingly, Faulhaber performs for a fixed sequence slightly better for values of z near to $\sum_{j=1}^{n} \frac{i_j}{2}$ than for values near 1 (see No. 16–17). There are two reasons for this behaviour: first, the binary search starts in the middle. Second, around $\sum_{j=1}^{n} \frac{i_j}{2}$, the polynomials often have a simpler structure than near 1 (cf. the final result from our elaborated example depicted in Figure 3) and can hence be evaluated faster.

Part	Time
Naïve	11.31 ms
Binomial Coefficients	1.18 ms
Bernoulli Numbers	5.21 ms
Faulhaber Coefficients	5.92 ms
Validity Intervals	0.08 ms
Variants of Faulhaber	
Sequential	18.32 ms
Sequential with Mirroring	14.02 ms
Parallel	6.61 ms
Parallel with Mirroring	5.56 ms

Table 2: Detailed run on No. 11. The parallel algorithm was executed with four threads.

#Threads	Time	Opt. Time	Ratio
1	46.993s	46.993s	100%
2	23.633s	23.326s	98.7%
3	16.355s	15.551s	95.1%
4	12.609s	11.663s	92.5%

Table 3: Run on No. 30 with a different number of threads. The optimal time (opt. time) is the time given by the division of the sequential running time by the number of threads.

- The test examples No. 18 and No. 19 are randomly generated examples with five and eight upper bounds, resp. In these cases Faulhaber performs much better. The running times for similar randomly generated instances behaved in a comparable way.

The parallel version of Faulhaber is additionally tested on those instances where the sequential version already had to spent some time on the computation. The measured times are listed in Table 1.

Table 2 gives a detailed view on the running times of the pre-computation phase, the computation of the validity intervals and some variants of Faulhaber, for the instance No. 11. The times for the variants of Faulhaber are without the pre-computation, and without computing the validity intervals. The times were measured internally, i.e., we did not measure the loading of the program. This is why the running times are slightly faster than the times measured in the other experiments.

Finally, we measured the speedup on No. 30 in Table 3. Up to the four available cores the speedup is nearly linear to the number of threads.

9 Application, Conclusion and Outlook

Application. An application of our algorithm is the study of computing the Pareto front of a discrete set in the field of multi-objective combinatorial optimization [12, 5]: Given n linear orders on a finite set U, the j-th linear or-

der $<_j$ induces an isomorphism ϕ_j from the elements of U to an interval $[1, i_j]$ such that $a <_j b$ iff $\phi_j(a) < \phi_j(b)$. We can assign each element $a \in U$ to a vector $(a_1, \ldots, a_n) \in \mathbb{N}_0^n$ such that $a_j = \phi_j(a) \in \mathbb{N}_0$ is the e_j-th element (we start counting from zero) with respect to the j-th criteria. The order on the vectors $u < v \Leftrightarrow_{df} \forall j : a_j \leq b_j$ and $\exists j : a_j < b_j$ induces the Pareto set $S = \{a \in U : \nexists b \in U : b < a\}$ of U. The order can be embedded in a lattice (adding missing elements such that the greatest and the least element are unique) whose Hasse diagram has the characteristic that every node $a \in S$ at depth d holds the property $\sum_{j=1}^{n} a_j = d$ (the top node has depth 0). Figure 6 depicts an exemplaric Hasse diagram where the a_i's are the bounds of our example from Section 4.1. Unfortunately, there is little known about the shape of this particular type of Hasse diagram. The problem of enumerating the nodes sharing the same depth is reducible to computing the number of ordered integer partitions with upper bounds, since the upper bounds are given by $(i_1, \ldots, i_n)$. The solutions described in this article can also be utilized to determine the number of depths z sharing the same maximal number of nodes in $\mathcal{O}(n \lg z)$ time by a binary search on the depths. Recent scenarios in the field of database research [20] present problems with more than five upper bounds whose sizes scale up to the big data domain of 10^5. In this domain, the Faulhaber based algorithm offers for the very first time the possibility for practical computation.

Conclusion. We have studied the computation of the number of different integer partitions with given bounds and a given integer value. To tackle this problem we presented a naïve algorithm and a novel data structure with a complex construction time. The data structure performed well on big instances (which are also of practical interest) whereas the naïve algorithm is preferable on simple small instances. We presented a parallel version of the construction algorithm of the proposed data structure. On some instances we could empirically measure that the speedup is nearly linear.

The main drawback of the presented data structure is the possibly exponential number of validity intervals, leading to the factor 2^n in the construction running time of $\mathcal{O}\big(2^n n^3\big)$. A substantial improvement of an approach based on validity intervals is hard to expect.

Outlook. To get a faster running time on small instances (e.g., the computation of the coefficients in Table 3 takes the majority of time of the complete execution), we could improve the computation of the coefficients as already mentioned in Section 5.

A totally different approach consists in exploiting results for discrete convolutions, since the equation from Lemma 3.2 is a discrete convolution of the functions $\begin{bmatrix}(i_1, i_2, \ldots, i_{n-m})\\ k\end{bmatrix}$ and $\begin{bmatrix}(i_{n-m+1}, i_{n-m+2}, \ldots, i_n)\\ z-k\end{bmatrix}$. Known algorithms for fast discrete convolution [9, 14] could improve the running times on large instances. Here further research is needed to investigate this approach.

Another way to parallelize the construction algorithm is to split the domain $[1, \sum_{j=1}^{n} i_j]$ of the final piecewise-defined polynomial up in π many intervals (the borders of the domain split are defined by borders of the validity intervals of the final piecewise-defined polynomial). The computation of a polynomial p is wait free if the polynomials of lower degree on which p depend are already

computed. To guarantee this, we let each thread additionally compute the lower degree polynomials needed by the final polynomial by themselves. This results in a higher work load, but perhaps a faster executing time since the execution is wait free.

Acknowledgements

This work is partly funded by the JSPS KAKENHI Grant Number JP18F18120.

References

[1] Andrews, G.E.: The Theory of Partitions. Cambridge University Press, first edn. (1998)

[2] Baker, Jr., H.C., Hewitt, C.: The incremental garbage collection of processes. SIGPLAN 12(8), 55–59 (1977)

[3] Beihoffer, D., Hendry, J., Nijenhuis, A., Wagon, S.: Faster algorithms for Frobenius numbers. The Electronic Journal of Combinatorics 12(27), 1–38 (2005)

[4] Carter, M.: Foundations of Mathematical Economics. MIT Press, first edn. (2001)

[5] Coello, C.A.C., Dhaenens, C., Jourdan, L.: Multi-objective combinatorial optimization: Problematic and context. In: Advances in Multi-Objective Nature Inspired Computing, Studies in Computational Intelligence, vol. 272, pp. 1–21. Springer (2010)

[6] Conway, H., Guy, R.: The Book of Numbers. Copernicus, first edn. (1995)

[7] Endres, M.: Semi-Skylines and Skyline Snippets - Theory and Applications. Books on Demand, first edn. (2011)

[8] Feller, W.: An Introduction to Probability Theory and Its Applications. Wiley publications in statistics, Chapman & Hall: London, second edn. (2008)

[9] von zur Gathen, J., Gerhard, J.: Modern Computer Algebra. Cambridge University Press, first edn. (2003)

[10] Glück, R., Köppl, D., Wirsching, G.: Computational aspects of ordered integer partition with upper bounds. In: Proc. SEA. LNCS, vol. 7933, pp. 79–90. Springer (2013)

[11] Hardy, G., Wright, E.: An Introduction to the Theory of Numbers. Oxford University Press, third edn. (1954)

[12] Klamroth, K.: Discrete multiobjective optimization. In: Proc. EMO. LNCS, vol. 5467, p. 4. Springer (2009)

[13] Knuth, D.E.: Johann Faulhaber and Sums of Powers. Math. Comp. 61(203), 277–294 (1993)

[14] Knuth, D.E.: Seminumerical Algorithms. Addison-Wesley, third edn. (1997)

[15] Landau, L.D., Lifshitz, E.M.: Quantum Mechanics–Non-relativistic Theory. Pergamon Press, second edn. (1965)

[16] Matoušek, J., Nešetřil, J.: Invitation to Discrete Mathematics. Oxford University Press, second edn. (2009)

[17] Park, G.: A generalization of multiple choice balls-into-bins: Tight bounds. Algorithmica 77(4), 1159–1193 (2017)

[18] Preisinger, T.: Graph-based algorithms for Pareto preference query evaluation. Ph.D. thesis, University of Augsburg, Germany (2009)

[19] Raab, M., Steger, A.: "Balls into bins" - A simple and tight analysis. In: Proc. RANDOM. LNCS, vol. 1518, pp. 159–170. Springer (1998)

[20] Wenzel, F., Köppl, D., Kießling, W.: Interactive toolbox for spatial-textual preference queries. In: Proc. SSTD. LNCS, vol. 8098, pp. 462–466. Springer (2013)

[21] Wirsching, G.: Balls in constrained urns and Cantor-like sets. Zeitschrift für Analysis und ihre Anwendungen 17, 979–996 (1998)

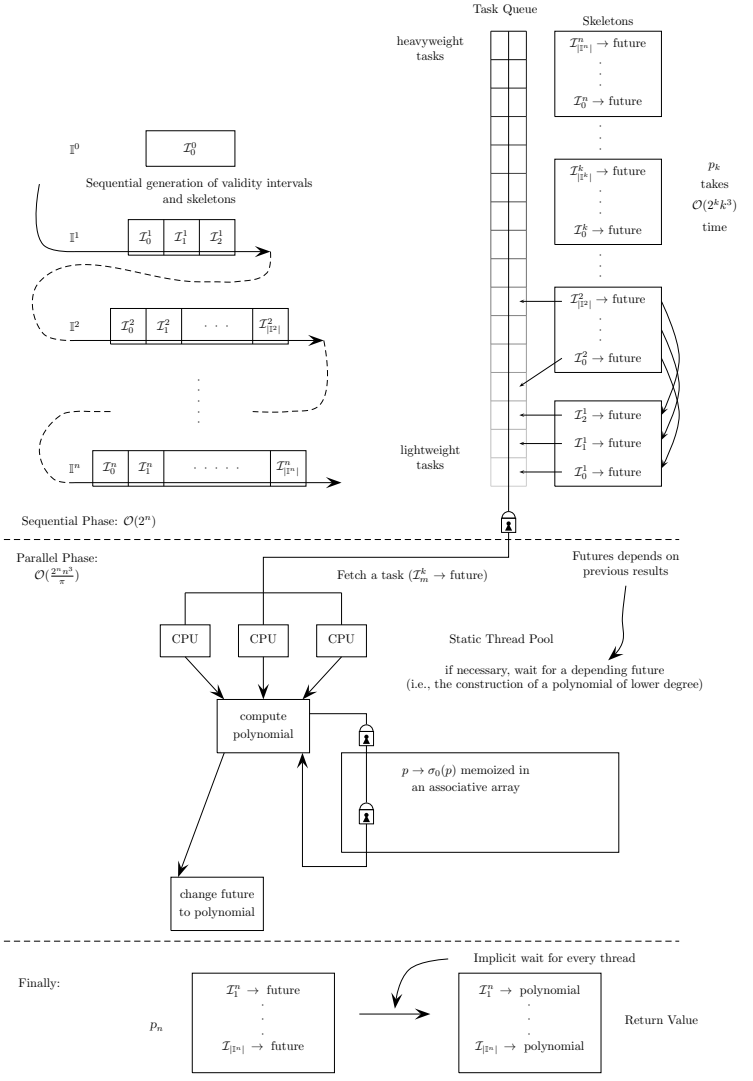

Figure 4: Schedule of the parallel algorithm. A preparing, sequential phase constructs the skeletons and pushes the futures (i.e., tasks described by the skeletons) in a queue. The futures are removed from the queue and processed by each thread in parallel. The computation finishes on transforming the n-th skeleton polynomial in a piecewise-defined polynomial.

No.	z	I	Faulhaber	Naïve
1	1	$(3, 4, 5)$	16	4
2	15	$(3, 4, 5)$	16	4
3	1	$(3000, 4000, 5000)$	15	3
4	6000	$(3000, 4000, 5000)$	16	7
5	1	$(10^4, 10^4, 10^4, 10^4, 10^4)$	15	2
6	300	$(10^4, 10^4, 10^4, 10^4, 10^4)$	15	19
7	$2.5 \cdot 10^4$	$(10^4, 10^4, 10^4, 10^4, 10^4)$	16	**113915**
8	25015	$(10000, 10005, 10010, 10015, 10020)$	17	**113991**
9	1	$(10993, 10520, 10856, 10346, 10039)$	17	4
10	26377	$(10993, 10520, 10856, 10346, 10039)$	17	**127070**
11	180	$(33, 29, 42, 34, 59, 76, 54, 33)$	30	16
12	40090	$(10000, 10005, 10010, 10015, 10021, 10027, 10039, 10063)$	92	**654035**
13	5000	$(10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000)$,	17	**>1 h**
14	52683	$(10993, 10520, 10856, 10346, 10039, 10644, 10005, 10941, 10718, 10305)$	763	**>1 h**
15	66131	$(12184, 12324, 14685, 11098, 13357, 13863, 10796, 10914, 10989, 11115, 10937)$,	2978	**>1 h**
16	1	$(12184, 12324, 14685, 11098, 13357, 13863, 10796, 10914, 10989, 11115, 10937, 13634)$	**15622**	2
17	72948	$(12184, 12324, 14685, 11098, 13357, 13863, 10796, 10914, 10989, 11115, 10937, 13634)$	15618	**>1 h**
18	2856	$(3696, 3894, 4137, 7588, 7816)$	16	**1508**
19	26433	$(5641, 9314, 969, 8643, 6291, 6241, 8747, 7041)$	70	**267207**
20	5000	$(1000^{\times 10})$	19	**>1 h**
21	5000	$(1000^{\times 20})$	51	**>1 h**
22	5000	$(1000^{\times 30})$	182	**>1 h**
23	20	J_5	16	2
24	20	J_6	19	5
25	20	J_7	38	4
26	20	J_8	119	2
27	20	J_9	523	3
28	20	J_{10}	**2384**	3
29	20	J_{11}	**10611**	4
30	20	J_{12}	**46993**	5
31	50	$(1^{\times 15}, 50)$	15	11
32	50	$(1^{\times 30}, 50)$	15	**108**

Figure 5: Experimental Results. The upper bounds $J_k = (j_0, \ldots, j_k)$ are defined by $j_1 = 3$, and $j_k = 2j_{k-1} + 1$ for $k > 1$. Time is in milliseconds. Especially bad times are visualized in red color and bold font. The notation $j^{\times k}$ means $j, \ldots, j$ (k many j's).

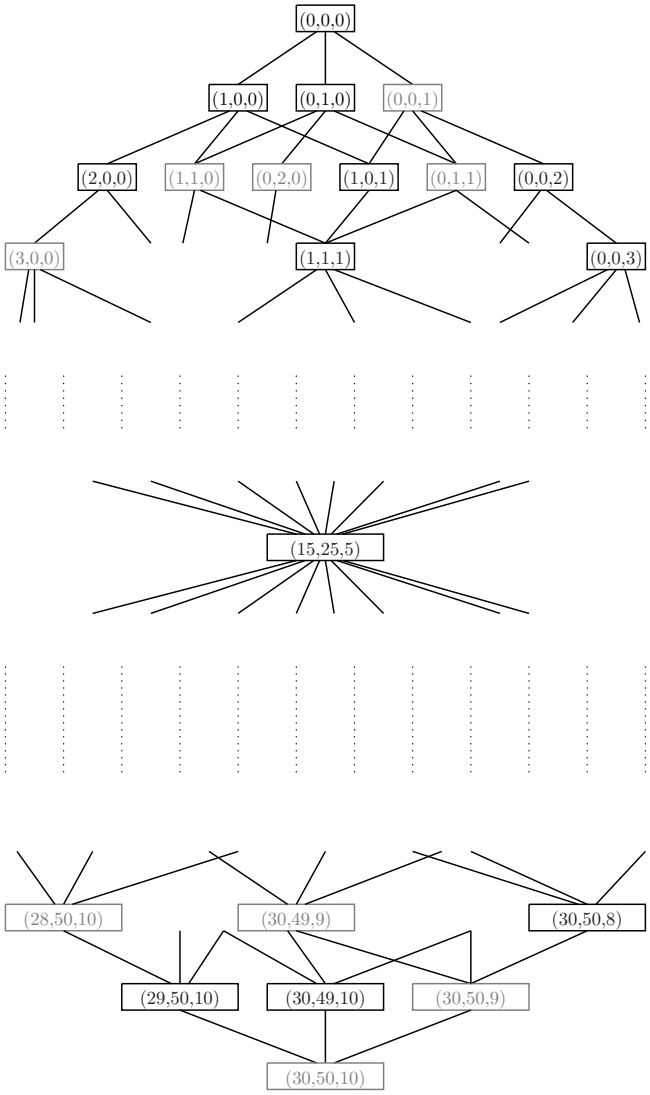

Figure 6: Hasse Diagram with the bounds $(30, 50, 10)$. The mapping of elements of U to the lattice is in general not surjective, i.e., there can be elements of the lattice that do not correspond to any element of the set U; we exemplaric grayed out some elements of the lattice to symbolize this fact.