Universiteit Utrecht

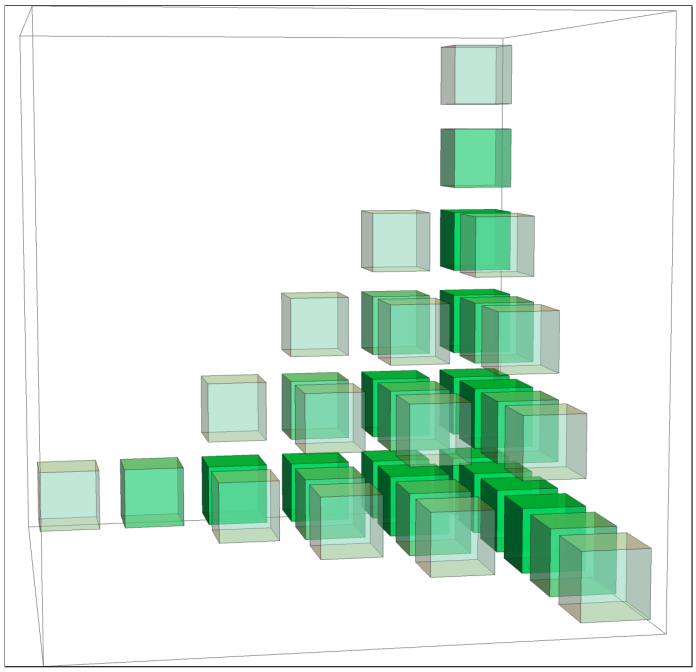
Faculteit Bètawetenschappen

Virtual counts of solid partitions

Bachelor Thesis

Frank Imbens

Mathematics



Supervisors:

Dr. M. Kool Supervisor
Mathematical Institute, Utrecht University

June 7 2019

Abstract

A partition is a finite descending sequence of non negative integers. This concept can be extended to multiple dimensions to form planar and solid partitions. For standard, or line, partitions there is an infinite product formula for the generating function by Euler which gives the number of partitions of a certain size. MacMahon found such a formula for planar partitions. However his conjecture for such an infinite product formula for solid partitions was incorrect for size 6 and up. In this thesis I will look at a conjecture made in an article by Y. Cao and M. Kool, giving a weighted generating function for solid partitions and verify it in many new cases.

Contents

1 Formal Power series

Before we can start to talk about partitions and their generating functions, we have to define the tools we use to talk about them, in particular formal power series which will form the generating functions. We will use the definition as given by in Generating Functionology by H. Wilf

Definition 1.0.1. We define the ring of formal power series of a ring R, $R[[q]]$, as the set of all infinite expressions:

$$f(q) = a_0 + a_1 q + a_2 q^2 + ... = \sum_{i \in \mathbb{N}} a_i q^i$$

This is a deceiving way of writing the sequence though. Because unlike the expansions of analytic or holomorfic functions, we are not concerned with convergence of the series. On these power series we define addition as:

$$\sum_{i \in \mathbb{N}} a_i q^i + \sum_{i \in \mathbb{N}} b_i q^i = \sum_{i \in \mathbb{N}} (a_i + b_i) q^i$$

And multiplication as:

$$\sum_{i \in \mathbb{N}} a_i q^i \sum_{i \in \mathbb{N}} a_i q^i = \sum_{i \in \mathbb{N}} \sum_{j \in \mathbb{N}} a_i b_j q^{i+j} = \sum_{i \in \mathbb{N}} (\sum_{j=0}^{i} a_j b_{i-j}) q^i$$

From now on we will only be using the ring $\mathbb{Z}$, and its formal power series.

Example 1.0.2. A well known sequence is the Fibonacci sequence:

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ F_{n+2} = F_{n+1} + F_n \end{cases}$$

Or:

$$(0, 1, 1, 2, 3, 5, 8, 13, ...)$$

Now we will attempt to calculate its generating function, as done in Generating Functionology, we first take its power series:

$$F(q) = \sum_{n \geq 0} F_n q^n$$

Now using how Fibonachi sequence is defined:

$$\sum_{n \geq 0} F_{n+1} q^n = \sum_{n \geq 0} F_n q^n + \sum_{n \geq 1} F_{n-1} q^n = F(q) + qF(q)$$

And also:

$$\sum_{n \geq 0} F_{n+1} q^n = (F_2 q + F_3 q^2 + ...) = \frac{F(q) - F_1 q}{q} = \frac{F(q) - q}{q}$$

So we remain with the equality:

$$F(q) + xF(q) = \frac{F(q)}{q} - 1$$

$$1 = \frac{F(q)}{q} - F(q) - xF(q)$$

$$q = F(q)(1 - q - q^2)$$

$$F(q) = \frac{q}{1 - q - q^2}$$

△

Later on we will create more complicated generating functions, specifically for the number of partitions. [Wil]

2 Line Partitions

Definition 2.0.1. A line partition is a descending sequence $(\pi_n)_{n\in\mathbb{N}}$ with each $\pi_n \in \mathbb{N}$, and only finitely many $\pi_n > 0$, the size of this partition is

$$|\pi| = \sum_{n\in\mathbb{N}} \pi_n$$

The simplest question we can ask about such partitions is how many there are of a certain size. To find an answer to this we try to find a nice expression for the generating function $f(q)$. This function is not a function in the normal sense of the word, but rather a way of writing to simplify a formal power series in a parameter q. We want this formal power series to be as follows:

$$f(q) = \sum_{\pi} q^{|\pi|} = \sum_{n\in\mathbb{N}} P_n q^n$$

In which P_n is the number of partitions of size n. Using the (formal) geometric series, and its generating function:

$$\sum_{n\in\mathbb{N}} q^n = \frac{1}{1-q}$$

Proposition 2.0.2. *We gain the following generating function for $P(n)$:*

$$\sum_{\pi} q^{|\pi|} = \prod_{k>0} \frac{1}{1-q^k}$$

Proof. The proof for this equality is quite simple. We start by expanding the right side of the formula into a product of formal power series:

$$\prod_{k>0} \frac{1}{1-q^k} = \prod_{k>0} (1 + q^k + q^{2k} + ...) = (1 + q + q^2 + ...)(1 + q^2 + q^4 + ...)(1 + q^3 + ...)...$$

Now we show a 1-1 correspondence between each term q^n that comes out of this product of formal power series, and a partition of size m. Each term q^m can be seen as a finite sequence in $\mathbb{N}$:

$$(n_1, n_2, n_3, ..., n_k) \equiv q^{1n_1} q^{2n_2} ... q^{kn_k} = q^{1n_1+2n_2+...+kn_k}$$

We link this to the partition that is $(k, .., k, .., 2..2, 1..1)$, with n_i times i, clearly this is a partition of size $1n_1 + 2n_2 + ... + kn_k$, this links one term of this product to one partition, and any partition is reached, and the partition of a term q^m is exactly size m, since this size $m = 1n_1 + 2n_2 + ... + kn_k$. Here we also use that for each partition, only finitely many terms are larger than 0, so in this product as that only finitely many terms are not $q^0 = 1$ □

3 Planar partitions

Definition 3.0.1. A planar partition is a sequence $(\pi_{i,j})_{i,j\in\mathbb{N}}$ such that, if $i \geq i'$ and $j \geq j'$ then $\pi_{i,j} \geq \pi_{i',j'}$, with $\pi_{i,j} > 0$ for only finitely many pairs (i, j). We again define the size of π as:

$$|\pi| = \sum_{i,j\in\mathbb{N}} \pi_{i,j}$$

Planar partitions too have a generating function. This is MacMahon's formula [Sta]:

$$\sum_{\pi} q^{|\pi|} = \prod_{k>0} \frac{1}{(1-q^k)^k}$$

4 Solid Partitions

Definition 4.0.1. A solid partition is a sequence in $(\pi_{i,j,k})_{i,j,k\in\mathbb{N}}$ such that, if $i \geq i'$, $j \geq j'$ and $k \geq k'$ then $\pi_{i,j,k} \geq \pi_{i',j',k'}$, with $\pi_{i,j,k} > 0$ for only finitely many tuples (i, j, k). We again define the size of π as:

$$|\pi| = \sum_{i,j,k\in\mathbb{N}} \pi_{i,j,k}$$

For solid partitions MacMahon also proposed a generating function:

$$\sum_{\pi} q^{|\pi|} = \prod_{k\in\mathbb{N}} \frac{1}{(1-q^k)^{\frac{1}{2}k(k+1)}}$$

However this identity fails at the level of q^6, giving $141q^6$ on the right hand side, while having $140q^6$ on the left hand side. An interesting result, since 6 can be argued to be the first size where 'real' solid partitions can happen. That is to say, up to the size 4 the only possible partitions are planar partitions embedded into a solid partition. They will have a maximal length of 1 in at least one direction. At size 5 the first partition appears that has at least length 2 in all directions, but this partition has values: $\pi_{1,1,1=2}$, $\pi_{2,1,1} = \pi_{1,2,1} = \pi_{1,1,2} = 1$ and $\pi_{i,j,k} = 0$ elsewhere. This partition is completely symmetric, at size 6 the first asymmetric partition arrives. Moreover the first completely asymmetrical solid partition appears, that is to say, the first partition which can have all $10 = 4!$ different orientations without any being the same. This partition is:

$$\begin{cases} \pi_{1,1,1} = 1 \\ \pi_{1,1,2} = \pi_{1,2,1} = \pi_{2,1,1} = 1 \\ \pi_{1,1,3} = 1 \\ \pi_{1,2,2} = 1 \\ \pi_{i,j,k} = 0 \text{ elsewhere} \end{cases}$$

Which can still be seen as a planar partition, since the height of this orientation is only 1. This partition looks as in Figure 1:

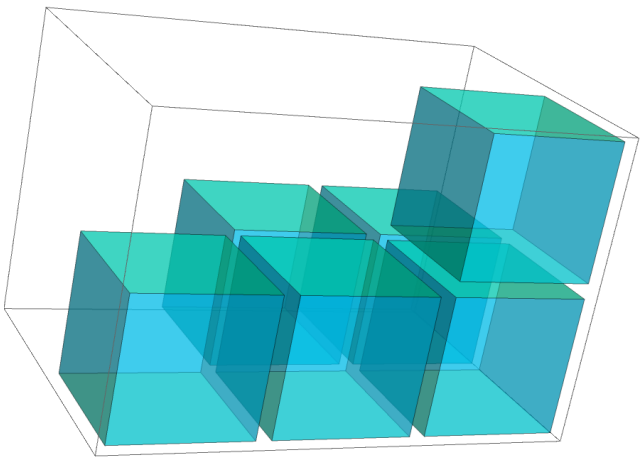


Figure 1: A partition of 6.

Now, some articles have already looked into finding a generating function of the type proposed by MacMahon, however an article by Donald Knuth [Knu70] provides an interesting insight. First he shows a method to

calculate the number of solid partitions. He also calculated the exponential $e(k)$, if we were to create a generating function of the kind

$$\sum_{\pi} q^{|\pi|} = \prod_{k \in \mathbb{N}} \frac{1}{(1-q^k)^{e(k)}}$$

These $e(n)$ go as follows:

$$0, 1, 3, 6, 10, 15, 20, 26, 34, 46, 68, 97, 120, 112, 23, -186, ...$$

For the first few entries this sequence follows a nice $\frac{1}{2}n(n+1)$ pattern, until $n = 6$, later on the series starts oscillating more and more wildly. This shows that this problem is much less simple than for line or planar partitions, meaning if we have to look for a generating function. We may have to look elsewhere for such a function.

4.1 Geometry and Algebra

Before we look at more generating functions, I would like to provide some background on where the idea comes from. This section will not go into the rigorous mathematics behind the conjecture, but simply provide a background for it.

We start with a Calabi-Yau manifold. This is a particular sort of smooth manifold. In our case the exact properties do not matter, we immediately specify that we will be using the simplest Calabi-Yau 4-fold that is $X = \mathbb{C}^4$. On this space we can define a group action by a torus. At first we look at the torus $(\mathbb{C}\backslash\{0\})^4 = (\mathbb{C}^*)^4$, with the action being defined as $(t_1, t_2, t_{3,4})(x_1, x_2, x_3, x_4) = (t_1x_1, t_2x_2, t_3x_3, t_4x_4)$. This clearly has only 1 fixed point.

A Hilbert scheme $\text{Hilb}^n(X)$ in our case is a set of ideals I in $\mathbb{C}[x_1, x_2, x_3, x_4]$ such that $\mathbb{C}[x_1, x_2, x_3, x_4]/I$ is a finite dimensional $\mathbb{C}$ vector space. Here the n represents the dimension of this vector space.

We can now define the very same action we defined on $\mathbb{C}^4$ on this space too. This is done by sending the monomial $x_1^{n_1}x_2^{n_2}x_3^{n_3}x_4^{n_4}$ to $(t_1x_1)^{n_1}(t_2x_2)^{n_2}(t_3x_3)^{n_3}(t_4x_4)^{n_4}$, and extending that linearly over each polynomial, and then to the ideal itself. Now the claim is that the fixed points of this action, $Hilb^n(X)^{(\mathbb{C}^*)^4}$, are only the ideals generated by monomials.

We now gain a very interesting representation of solid partitions. For simplicity we will first look at the 2 dimensional case, this gives line partitions instead of solid partitions, but is much easier to visualize. We take the ideal generated by $\{x_1^3, x_1^2x_2, x_2^2\}$, this can be visualised as in Figure 2:

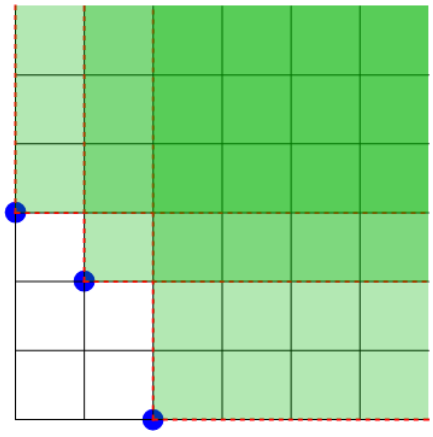


Figure 2: An ideal visualised as line partition, the vertical axis is powers of x_1, the horizontal axis is powers of x_2.

We use only minimal generators of the ideal here. As seen the basis of $\mathbb{C}[x_1, x_2]/I$ appears, these are the points not in or bordering on the green area. Together these form a line partition. The same happens for solid partitions in the 4-dimensional case. So there seems to exist a 1-1 correspondence between the fixed points of this Hilbert Scheme $Hilb^n(X)^{(\mathbb{C}^*)^4}$ and partitions of size n.

The theory of these Calabi-Yau 4-folds is applied mostly in physics in string theory. A recent example is Magnificent Four with Color[NP19]. However such applications are beyond the scope of this thesis.

4.2 Weights

To try and cast more light on the generating function of solid partitions we define a few functions for each partition, we define these functions like in [CK18]. In this article each of the following polynomials are based on an interpretation in geometry, however here I will not go into the geometry, merely the combinatorial conjecture of this article.

Definition 4.2.1. We define:

$$Z_\pi = \sum_{i,j,k>0} \sum_{l>0}^{\pi_{i,j,k}} t_1^{i-1} t_2^{j-1} t_3^{k-1} t_4^{l-1}$$

This polynomial is in essence a term for every 4-dimensional cube on the lattice, with the exponentials as its coordinates. This is a different way of representing a solid partition than the standard 3 dimensional sequence of positive integers.

Definition 4.2.2. We define an operation $\overline{(\cdot)}$ on such polynomials as follows

$$\overline{t_1^{n_1} t_2^{n_2} t_3^{n_3} t_4^{n_4}} = t_1^{-n_1} t_2^{-n_2} t_3^{-n_3} t_4^{-n_4}$$

We extend this ($\mathbb{Z}$-)linearly over these polynomials. This can also be seen as: $\overline{Z_\pi}(t_1, t_2, t_3, t_4) = Z_\pi(t_1^{-1}, t_2^{-1}, t_3^{-1}, t_4^{-1})$.

Definition 4.2.3. This map is used to define a new rational function:

$$V_\pi = Z_\pi + \frac{\overline{Z_\pi}}{t_1 t_2 t_3 t_4} - \frac{Z_\pi \overline{Z_\pi}(1-t_1)(1-t_2)(1-t_3)(1-t_4))}{t_1 t_2 t_3 t_4}$$

We create a degeneracy in these polynomials by enforcing the equality $t_1 t_2 t_3 t_4 = 1$.

Definition 4.2.4. We now define a new operation on these polynomials:

$$e_T(\sum_a a_0 t_1^{a_1} t_2^{a_2} t_3^{a_3} t_4^{a_4}) = \prod_a (a_1 l_1 + a_2 l_2 + a_3 l_3 + a_4 l_4)^{a_0}$$

The degeneracy created by $t_1 t_2 t_3 t_4 = 1$ now becomes $l_1 + l_2 + l_3 + l_4 = 0$, in essence we have switched multiplication and addition, and so have their unit elements.

Proposition 4.2.5. *For any partition we can see that each term in $e_T(-V_\pi)$ appears twice, once with a positive and once with a negative sign.*

Proof. We prove that each of the $t_1^{a_1} t_2^{a_2} t_3^{a_3} t_4^{a_4}$ terms appear exactly twice in the sum V_π, once with negative and once with positive power.We do this since it is equivalent, and easier. We see from the definition that:

$$V_\pi = Z_\pi + \frac{\overline{Z_\pi}}{t_1 t_2 t_3 t_4} - \frac{Z_\pi \overline{Z_\pi}(1-t_1)(1-t_2)(1-t_3)(1-t_4))}{t_1 t_2 t_3 t_4}$$

Where:

$$(1-t_1)(1-t_2)(1-t_3)(1-t_4) = 1 - \sum_i t_i + \sum_{i \neq j} t_i t_j - \sum_{i \neq j \neq k} t_i t_j t_k + t_1 t_2 t_3 t_4$$

Using that $t_1 t_2 t_3 t_4 = 1$. So now V_π becomes:

$$V_\pi = Z_\pi + \overline{Z_\pi} + f(t_1, t_2, t_3, t_4) Z_\pi \overline{Z_\pi}$$

Where:

$$f(t_1, t_2, t_3, t_4) := 2 - \sum_i (t_i + t_i^{-1}) + (t_1 t_2 + t_1^{-1} t_2^{-1} + t_1 t_3 + t_1^{-1} t_3^{-1} + t_1 t_4 + t_1^{-1} t_4^{-1})$$

Since the terms $t_1^{n_1} t_2^{n_2} t_3^{n_3} t_4^{n_4}$ and $t_1^{-n_1} t_2^{-n_2} t_3^{-n_3} t_4^{-n_4}$ in Z_π and $\overline{Z_\pi}$ form pairs, so do the terms in their product. Here the other part of each pair is the product of the two opposing terms. The same holds for f and $Z_\pi \overline{Z_\pi}$. For instance if we have a term ab^{-1} then we have a term ba^{-1}. Since $f(t_1, t_2, t_3, t_4)$ has both a term a and a^{-1} and $Z_\pi \overline{Z_\pi}$ has a term b and b^{-1}. Since this pairing carries over through e_T, it also applies to $e_T(-V_\pi)$, with negative signs instead of negative powers. □

Definition 4.2.6. We now define a last polynomial

$$L_\pi(d_1,d_2,d_3,d_4) := e_T(t_1^{d_1}t_2^{d_2 1}t_3^{d_3}t_4^{d_4}Z_\pi)$$

Expanding this expression, using the definition of e_T we get:

$$L_\pi(d_1,d_2,d_3,d_4) = \prod_{i,j,k\in\mathbb{N}}\prod_{m=1}^{\pi_{i,j,k}}((d_1+i-1)l_1+(d_2+j-1)l_2+(d_3+k-1)l_3+(d_4+m-1)l_4)$$

Example 4.2.7. Suppose we have a simple partition, namely $\pi_{1,1,1}=1$, $\pi_{2,1,1}=1$, with $\pi_{i,j,k}=0$ everywhere else, a partition of size 2 (a partition of size 1 does gives a constant polynomial). Calculating all the defined polynomials gives:

$$Z_\pi = 1+t_1$$

and:

$$V_\pi = 1+t_1+\frac{1+t_1^{-1}}{t_1t_2t_3t_4}-\frac{(1+t_1)(1+t_1^{-1})(1-t_1)(1-t_2)(1-t_3)(1-t_4)}{t_1t_2t_3t_4} =$$

Already using that $t_1t_2t_3t_4=1$, thus eliminating t_4

$$2+t_1+t_1^{-1}-(-t_1^2t_2t_3-t_1^{-2}t_2^{-1}t_3^{-1}+t_1^2t_2+t_1^{-2}t_2^{-1}+t_2^2t_3+t_1^{-2}t_3^{-1}-t_1^2-t_1^{-2}-t_1t_2t_3-t_1^{-1}t_2^{-1}t_3^{-1}+t_1t_2^{-1}t_3^{-1}+t_1^{-1}t_2t_3$$
$$+t_1t_2+t_1^{-1}t_2^{-1}-t_1t_2^{-1}-t_1^{-1}t_2+t_1t_3+t_1^{-1}t_3^{-1}-t_1t_3^{-1}-t_1^{-1}t_3-t_2-t_2^{-1}-t_3-t_3^{-1}+2)$$

And:

$$L_\pi = e_T(t_1^{d_1}t_2^{d_2}t_3^{d_3}t_4^{d_4}(1-t_1))$$

△

Having defined all these polynomials, we now look at the conjecture in [CK18]:

Conjecture 4.2.8. *(Cao-Kool)*

$$\sum_\pi w_\pi L_\pi(d_1,d_2,d_3,d_4)q^{|\pi|} = M(q)^{\frac{(d_1l_1+d_2l_2+d_3l_3+d_4l_4)(-l_1l_2l_3-l_1l_2l_4-l_1l_3l_4-l_2l_3l_4)}{l_1l_2l_3l_4}}$$

Where we define $w_\pi = \pm\sqrt{(-1)^{|\pi|}e_T(-V_\pi)}$ where $(-1)^{|\pi|}$ under the root is to make sure the term under the square root is positive. Also w_π is defined up to it's sign, where a choice must be made to satisfy the equality, the conjecture states this choice is unique, and exists. Remember that each term in the product of V_π appears twice, with opposing signs, so this term is of the same sort as L_π, a product of sums of the four variables.
We also remember the definition of MacMahon's formula:

$$M(q) = \prod_{k>0}\frac{1}{(1-q^k)^k}$$

We will however specialize this formula, by setting $d_1=d_2=d_3=0$ and adding a new variable $-d=d_4$, as seen in Remark 3.24 in [CK18]and also that $l_1+l_2+l_3=0$ this leads to the following formula:

Conjecture 4.2.9. *(Cao-Kool)*

$$\sum_\pi w_\pi L_\pi(0,0,0,-d)q^{|\pi|} = M(q)^d$$

In this equation $w_\pi L_\pi(0,0,0,-d)$ is said to be of the shape $\omega_\pi^{\mathrm{DT}}\prod_{l=1}^{\pi_{1,1,1}}(d-(l-1))$. However, a yet unpublished result by J. Rennemo [Ren19] shows this to be false. Instead this polynomial in d becomes of the form $\omega_\pi\prod_{(i,l)|1\le l\le\pi_{i,i,i}}(d-(l-i))$. But before we see the result of this difference, we first look at another possible weight. This same result by Rennemo states that there are no poles in $l_1+l_2+l_3$ for the previous formulas, meaning that the specialization that sets $l_1+l_2+l_3=0$ is allowed.

4.3 The classical weight

Although there is no proof yet for the equality given before, we can prove another equality, for different weights.

$$\sum_{\pi} \omega_{\pi}^{\mathrm{cl}}(\prod_{l=1}^{\pi_{1,1,1}} (d-(l-1)))q^{|\pi|} = M(q)^d$$

Here $\pi_{1,1,1}$ is the height of the partition, and $\omega_{\pi}^{\mathrm{cl}}$ is the classical weight. We define $\omega_{\pi}^{\mathrm{cl}}$ using plane-partitions. To do this we first have to make sense of "stacking" these planar partitions. We do this by turning a planar partition into a solid partition of height 1:

$$\pi_{i,j,k} = \begin{cases} 0, & \text{if } k \leq \pi_{i,j} \\ 1, & \text{if } k > \pi_{i,j} \end{cases}$$

after which we can simply add the partitions pointwise. That we indeed get a partition is because since we start with planar partitions, the flat solid partition we get is also a partition. After addition this remains a partition, a finite descending sequence in 3 directions. This is in essence a partition of partitions.
Now to get $\omega_{\pi}^{\mathrm{cl}}$ we sum over all possible ways a partition can be build up out of such flat partitions, and then divide each of these parts of the sum by $m_1!...m_k!$ for m_i the number of times we use the i-th flat partition. We can now prove the following equality, which we will use to prove the earlier formula, where $h(\pi) = \pi_{1,1,1}$ the height of the partition[CK18].

Proposition 4.3.1. *[CK18]*

$$\sum_{\pi} \omega_{\pi}^{\mathrm{cl}} t^{h(\pi)} q^{|\pi|} = e^{t(M(q)-1)}$$

Proof. We see:

$$e^{t(M(q)-1)} = e^{t(\sum_{\{\pi_{i,j}\}}(q^{|\pi_{i,j}|})-1)} = \prod_{\pi_{i,j}} e^{tq^{|\pi|}}$$

Here we exclude the flat partition that is empty. Expanding the exponential we get:

$$\prod_{\pi_{i,j}} \sum_{n=0}^{\infty} \frac{(tq^{|\pi|})^n}{n!} = \prod_{\pi_{i,j}} (1 + tq^{|\pi|} + \frac{tq^{|\pi|}}{2!} + ...)$$

Using the same trick as for line partitions, each possible cluster in this product becomes a solid partition, divided by the same product of factorials as $\omega_{\pi}^{\mathrm{cl}}$. Furthermore the power of t in such a cluster is the amount of flat partitions we add. Since we exclude the empty partition this is also the height $\pi_{1,1,1}$ so this equals:

$$\sum_{\pi_{i,j,k}} \omega_{\pi}^{\mathrm{cl}} t^{\pi_{1,1,1}} q^{|\pi|}$$

□

We now use this expression to prove the earlier expression:

Proposition 4.3.2. *[CK18]*

$$\sum_{\pi} \omega_{\pi}^{\mathrm{cl}}(\prod_{l=1}^{\pi_{1,1,1}} (d-(l-1)))q^{|\pi|} = M(q)^d$$

Proof. We will prove this as done in [CK18], however for us the prove will work in reverse, since we start with the equality with an exponential.

$$\sum_{\pi} \omega_{\pi}^{\mathrm{cl}} t^{h(\pi)} q^{|\pi|} = e^{t(M(q)-1)}$$

We split the exponential:

$$\sum_{\pi} \omega_{\pi}^{\mathrm{cl}} t^{h(\pi)} q^{|\pi|} = \sum_{k} t^k \frac{(M(q)-1)^k}{k!}$$

Splitting by powers of t we get the equality:

$$\sum_{\pi_{1,1,1}=k} \omega_\pi^{\mathrm{cl}} t^k q^{|\pi|} = t^k \frac{(M(q)-1)^k}{k!}$$

Expanding using binomial coefficients:

$$\sum_{\pi_{1,1,1}=k} \omega_\pi^{\mathrm{cl}} t^k q^{|\pi|} = t^k \frac{\sum_{i=1}^{k} \frac{k!}{(k-i)!i!}(-1)^{k-i} M(q)^i}{k!}$$

$$= t^k \sum_{i=1}^{k} \frac{1}{(k-i)!i!}(-1)^{k-i} M(q)^i$$

We now sum the terms up to k:

$$t^k \sum_{\pi_{1,1,1}=k} \omega_\pi^{\mathrm{cl}} q^{|\pi|} = t^k \sum_{i=1}^{k} \frac{1}{(k-i)!}(-1)^{k-i} \frac{M(q)^i}{i!}$$

To get:

$$\frac{M(q)^k}{k!} = \sum_{i=1}^{k} \sum_{\pi_{1,1,1}=i} \omega_\pi q^{|\pi|} \frac{1}{(k-i)!}$$

This is done by solving a system of k linear equations for $M(q)^k$ Now for $d = k$ and a partition of height i, $\frac{k!}{(k-i)!}$ is the same as $\prod_{l=1}^{\pi_{1,1,1}}(d-(l-1)))$, so:

$$\sum_{\pi} \omega_\pi^{\mathrm{cl}} \Big(\prod_{l=1}^{\pi_{1,1,1}} (d-(l-1))) q^{|\pi|} = M(q)^d$$

□

4.4 The classical weight and DT weight

Now we try to compare these two weights. First we have to show they are not the same. Interestingly this happens at size 8. This is the first size where a completely asymmetric partition appears, that is not also a plane partition (if rotated correctly). An example of such a partition is:

$$\begin{cases} \pi_{1,1,1} = 3 \\ \pi_{1,1,2} = \pi_{1,2,1} = \pi_{2,1,1} = 1 \\ \pi_{1,1,3} = 1 \\ \pi_{1,2,2} = 1 \\ \pi_{i,j,k} = 0 \text{ elsewhere} \end{cases}$$

This partition is different in all $4! = 10$ different orientations, since each of the 4 directions is different. First it splits based on $\pi_{1,2,2}$ which the side is either attached to or not, and then those 2 groups of 2 are split by having length 2 or 3. Interestingly this is not the case where the classical and DT weights are different, the partition where this happens is (as seen in Figure 3):

$$\begin{cases} \pi_{1,1,1} = 1 \\ \pi_{1,1,2} = 1 \\ \pi_{1,2,1} = 1 \\ \pi_{1,2,2} = 1 \\ \pi_{2,2,1} = 1 \\ \pi_{1,2,2} = 1 \\ \pi_{2,1,2} = 1 \\ \pi_{2,2,2} = 1 \\ \pi_{i,j,k} = 0 \text{ elsewhere} \end{cases} \tag{1}$$

Which has $\omega_\pi^{\text{cl}} = 1$, giving a weight of d. The DT-weight is $\frac{1}{2}d(d+1)$. This is the first partition where

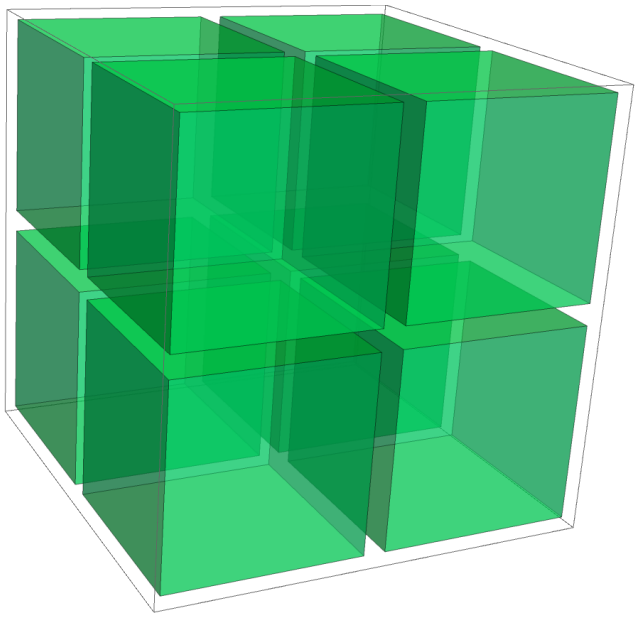


Figure 3: Een partitie van 8, met $\pi_{2,2,2} = 1$.

$\pi_{2,2,2} > 0$, which ties into the result by [Ren19]. This result states that for the specialization $l_1 + l_2 + l_3 = 0$, we do not gain

$$w_\pi L_\pi(0,0,0,-d) = \omega_\pi \prod_{l=1}^{\pi_{1,1,1}} (d-(l-1))$$

but rather:

$$w_\pi L_\pi(0,0,0,-d) = \omega_\pi \prod_{(i,l)|1\leq l\leq \pi_{i,i,i}} (d-(l-i))$$

Clearly this has an impact on this difference, since you get a second order polynomial in d instead of a first order one. One may think this leads to a counterexample of CK's conjecture. However there is exactly one other solid partition of size 8 with a different weight:

$$\begin{cases} \pi_{1,1,1} = 2 \\ \pi_{1,1,2} = 1 \\ \pi_{1,2,1} = 1 \\ \pi_{1,2,2} = 1 \\ \pi_{2,2,1} = 1 \\ \pi_{1,2,2} = 1 \\ \pi_{2,1,2} = 1 \\ \pi_{2,2,2} = 0 \\ \pi_{i,j,k} = 0 \text{ elsewhere} \end{cases} \tag{2}$$

This has DT-weight: $\frac{1}{2}d(d-1)$ and classical weight $d(d-1)$. The sum of both these pairs is d^2 meaning that the differences cancel out. If the conjecture is true that would mean that for each order q^n such cancellations would happen.

5 Computer calculations

In this section I will attempt to calculate the DT-weights of all partitions of a certain size, to verify certain orders q^n of the conjecture in [CK18].

5.1 Generating solid partitions

As was commented in [Knu70], calculating the number of solid partitions of a certain size is no easy problem. The current highest size for which the number is calculated is 72 [OEI]. Of course one could try calculate the number of partitions by brute force, running a loop over all 3 dimensional arrays of size n with integers between 0 and n, however this would quickly go bad, since there are $n^{(n)^3}$ options. So we must find a more elegant and efficient way to calculate this number.

For this, we first create an order of walking through the n^3 list we have created, by increasing the last coordinate until we reach the edge, and then increasing the second to last coordinate by a step, until that reaches the edge, and then increasing the first coordinate by a step. in a $2 * 2 * 2$ list this would go as:

$$(0,0,0) \to (0,0,1) \to (0,1,0) \to (0,1,1) \to (1,0,0) \to (1,0,1) \to (1,1,0) \to (1,1,1)$$

Since the size is 2 here, this is like counting in a binary system. For a partition of size n it's like counting in an n-size system. Now, when we reach a certain position, it can have a few possible heights, anywhere from 0 to n. However this height must be lower than the difference between total size already gained. It must also be lower than the height of each of the spots with one coordinate being lowered by 1, so for (i,j,k) that would be $(i-1,j,k)$, $(i,j-1,k)$ and $(i,j,k-1)$. Since we reach all the spots influencing a certain position before reaching that position, this means we simply take the minimum of those 3 numbers, and the number we still need to reach the wanted size. Then the program then simply loops over each of the possible sizes, and continues to the next step by a recursive function.

This algorithm will still be quite slow, since it will still have to complete a full recursion stack even if it has already filled a partition. Furthermore it has to check whether it is actually a partition of n at the end, and not a lower valued one. So we employ a few methods to throw away useless calculation/stack creation. This is often called pruning in algorithms, since the recursion is much like a tree (a type of graph), with each new stack forming a vertex, and each possible value looped over an edge to a next stack.

The main method depends on the possible heights of the current position. If the only possible height for the position is 0, that can be for two reasons. The first being you are out of size, the partition size is already reached. In this case we can simply cut off the entire rest of the branch, since all that branch would be doing would be creating a long recursion stack of 0's.

The next option if the current maximum height is 0 is that it is limited by the heights that are respectively one coordinate lower in i, j or k direction. Here we have a few options, increasing in performance gain: Without further specifications on coordinates, we can skip the entire rest of the line in the k direction (assuming that is the first coordinate we iterate over), so we go from (i,j,k) to $(i,j+1,k)$. See Figure 4.
However, if the k coordinate is already 0, we are limited by the height of the partition one back in the i or j direction. So we can instantly skip an entire plane, going from $(i,j,0)$ to $(i+1,0,0)$. See Figure 5.
And lastly, if both the k and j coordinates are already 0, we are limited by a height one down in the i direction. This means it will be impossible to add anymore cubes, since everything in the plane below this (in the i direction) must already be 0. So we can cut off the entire branch here too, see Figure 6.
Having made these additions, the program is much faster, allowing for the calculation of the amount of solid partitions up to size 20, in around 2 minutes. The following list was calculated, this is seen in Figure 7.
Which is is in agreement with [**OEIS**].

5.2 Calculating the classical weights

As defined before, given a partition π, the classical weight is all possible ways to create the partition as a sum of partitions of height one, each times a factor dependent on the amount of each partition in the sum.

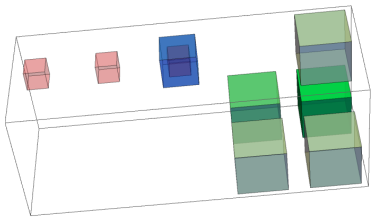


Figure 4: How we can skip an entire line. The k direction is left, j is towards the reader, i is up. The blue cube is the current location, the red cubes is what can be skipped and the green cubes form the partition, with the saturation and opacity showing the height at a position.

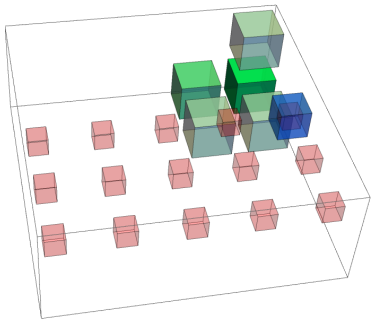


Figure 5: How we can skip an entire Plane. The k direction is left, j is towards the reader, i is up. The blue cube is the current location, the red cubes is what can be skipped and the green cubes form the partition, with the saturation and opacity showing the height at a position.

Now, given a solid partition we want a program to calculate the classical weight for it.

To do this, we first create a file containing all partitions of height 1 up to a certain size, this is saved in a file.These partitions are in essence planar partitions. This file is made so instead of calculation all these partitions when the script starts they are already generated, saving a lot of calculation time. Now, given a partition, this file is loaded, and every planar partition in it that is smaller than or equal to the partition in every point is put in a list. We only need these partitions, since if any other partition cannot be part of a sum that totals the given partition. Removing them will significantly decrease the calculation time.

Now we loop over this list, using recursion. Adding each partition a number of times, in the same way as we did to calculate the partitions. For each partition we have n possible outcomes, with n the height difference between the given partition and the summed partition. Once the correct height is reached, we add $\frac{1}{m_1!\ldots m_k!}$ to the total weight if the summed partition is equal to the given partition. Where m_i is the amount of times the i-th partition was added to the sum. To speed up the calculations further, any time a new partition is added to the sum the script checks if the summed partition is still smaller than the given partition (in each point), if not the calculation is stopped, and we return to the previous level of the recursion.

5.3 Calculating the polynomial DT-weight

For the calculation of the DT-weight, Mathematica was used. Firstly we import all solid partitions of a certain size, which are generated by the earlier python script. These imported lists are then turned into

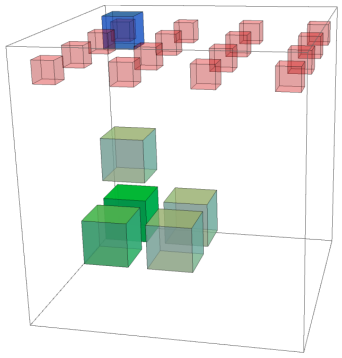


Figure 6: How we can skip the rest of the recursion. The k direction is towards the reader, j is to the right, i is up. The blue cube is the current location, the red cubes is what can be skipped and the green cubes form the partition, with the saturation and opacity showing the height at a position.

polynomial expressions, which are turned into pure functions. The script uses pure functions, as they are easier to create in large quantities like we have with a large list of polynomial expressions. So this leaves the script with a list of polynomials Z_π for each partition π of a certain size.

After this a function is mapped over this list, that first calculates V_π, and then L_π and w_π. After this it simplifies the expression, takes the specialization $l3 = -l1 - l2$. The final polynomial expression in d (note that this is no longer a function) is printed together with Z_π and it is returned. So now the script has a list of all weights $w_\pi L_\pi$. Note that we take the sign of w_π so that the coefficient of the highest power of d is positive. It is conjectured in [CK18] that this will yield the right choice of signs. The total sum of the list is taken, and all terms are expanded, this will yield a polynomial in d. This should equal the coefficient of q^n in the expression $M(q)^d$ with n the size of the partitions imported. Like this we have checked order q^7:

Order	Coefficient
q^0	1
q^1	d
q^2	$\frac{d^2}{2} + \frac{5d}{2}$
q^3	$\frac{d^3}{6} + \frac{5d^2}{2} + \frac{10d}{3}$
q^4	$\frac{d^4}{24} + \frac{5d^3}{4} + \frac{155d^2}{24} + \frac{21d}{4}$
q^5	$\frac{d^5}{120} + \frac{5d^4}{12} + \frac{115d^3}{24} + \frac{163d^2}{12} + \frac{26d}{5}$
q^6	$\frac{d^6}{720} + \frac{5d^5}{48} + \frac{305d^4}{144} + \frac{217d^3}{16} + \frac{8597d^2}{360} + \frac{25d}{3}$
q^7	$\frac{d^7}{5040} + \frac{d^6}{48} + \frac{95d^5}{144} + \frac{367d^4}{48} + \frac{11411d^3}{360} + \frac{233d^2}{6} + \frac{50d}{7}$

All these coefficients are the same as in $M(q)^d$, shown below (up to order q^{10}):

$$1$$

$$d$$

$$\frac{d^2}{2} + \frac{5d}{2}$$

$$\frac{d^3}{6} + \frac{5d^2}{2} + \frac{10d}{3}$$

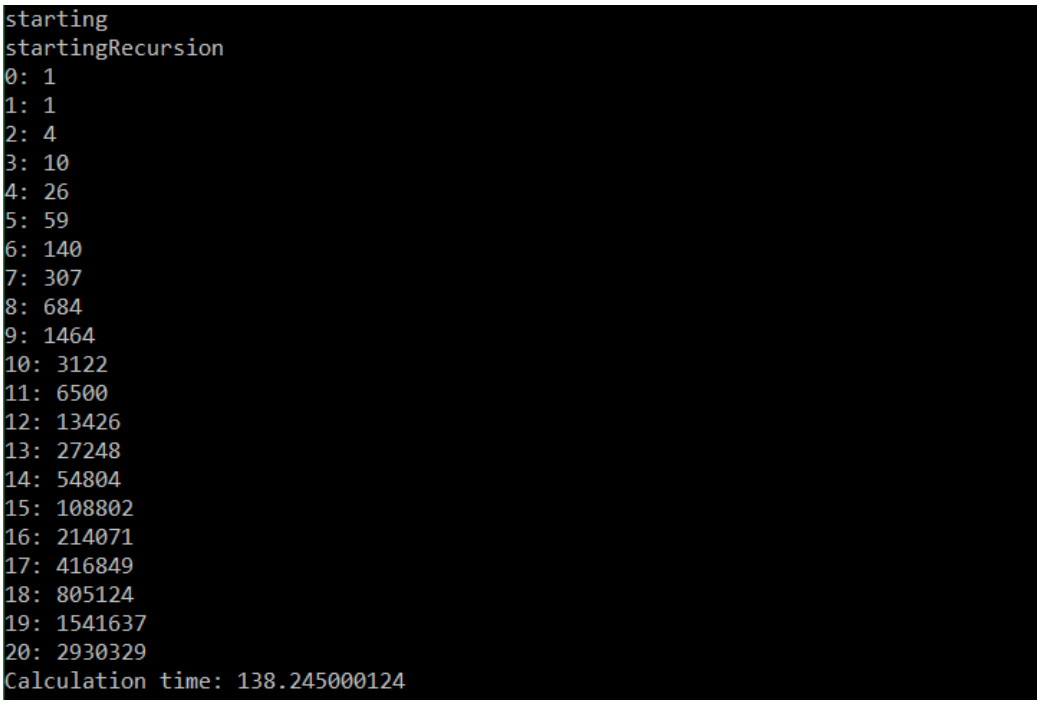


Figure 7: The calculated amount of solid partitions of each size.

$$\frac{d^4}{24}+\frac{5d^3}{4}+\frac{155d^2}{24}+\frac{21d}{4}$$

$$\frac{d^5}{120}+\frac{5d^4}{12}+\frac{115d^3}{24}+\frac{163d^2}{12}+\frac{26d}{5}$$

$$\frac{d^6}{720}+\frac{5d^5}{48}+\frac{305d^4}{144}+\frac{217d^3}{16}+\frac{8597d^2}{360}+\frac{25d}{3}$$

$$\frac{d^7}{5040}+\frac{d^6}{48}+\frac{95d^5}{144}+\frac{367d^4}{48}+\frac{11411d^3}{360}+\frac{233d^2}{6}+\frac{50d}{7}$$

$$\frac{d^8}{40320}+\frac{d^7}{288}+\frac{91d^6}{576}+\frac{419d^5}{144}+\frac{128167d^4}{5760}+\frac{18709d^3}{288}+\frac{39709d^2}{672}+\frac{85d}{8}$$

$$\frac{d^9}{362880}+\frac{d^8}{2016}+\frac{53d^7}{1728}+\frac{33d^6}{40}+\frac{175669d^5}{17280}+\frac{5339d^4}{96}+\frac{2207983d^3}{18144}+\frac{210571d^2}{2520}+\frac{91d}{9}$$

$$\frac{d^{10}}{3628800}+\frac{d^9}{16128}+\frac{121d^8}{24192}+\frac{1067d^7}{5760}+\frac{582613d^6}{172800}+\frac{69287d^5}{2304}+\frac{1128287d^4}{9072}+\frac{4258643d^3}{20160}+\frac{2967379d^2}{25200}+13d$$

However the used mathematica program is rather slow, taking multiple seconds per polynomial at size 7. Calculations of all weights took more than 60 minutes, making it unfeasible to continue. So a more efficient

algorithm was made. This does not use the symbolic calculations of mathematica, but rather a newly made object in C, which represents the polynomials. This object consists of a multi-dimensional list, which forms the polynomial (with negative powers), and the location of the constant term in this list. On this multiplication, addition and subtraction are defined. Then a function is made to transform it to the product of sums of l_i. This is done in essence by not changing the object, but rather changing what multiplication means on the object. A method is made to simplify the term, such as $\frac{(l_1+l_2+l_3)}{(-l_1-l_2-l_3)} = -1$, after which the specialisations are applied, firstly $l_1+l_2+l_3+l_4 = 0$, and then $l_1+l_2+l_3 = 0$. Instead of doing this entire calculation with d remaining a variable, it is calculated for multiple (in this case integer) values of d. After which these values are all added up. We are left with 26, namely 1 through to 26, points the polynomial intersects, allowing us to reconstruct the polynomial in d, assuming its order is 25 or lower. That this is the case follows from [Ren19]'s result. This polynomial in d is now the coefficient of q^k. Using this algorithm the coefficients up to q^{25} were checked. These will be included in the appendix C.

5.4 An example

Here I will also calculate the weight of a single partition of size 20. The partition used as an example is given by:

$$Z_\pi = 1+t_1+t_1^2+t_2+t_1t_2+t_1^2t_2+t_3+t_1t_3+t_2t_3+t_1t_2t_3+t_3^3+t_4+t_1t_4+t_2t_4+t_3t_4+t_2t_3t_4+t_4^2+t_2t_4^2+t_4^3$$

We visualise it as in Figure 8. The calculated weight for this partition is $\frac{6}{5}d\left(d^4-5d^3+5d^2+5d-6\right)$. This is $\frac{6}{5}d(d-1)(d-2)(d-3)(d+1)$ which is

$$\omega_\pi \prod_{(i,l)|1\leq l\leq \pi_{i,i,i}} (d-(l-i))$$

For $\omega_\pi = \frac{6}{5}$

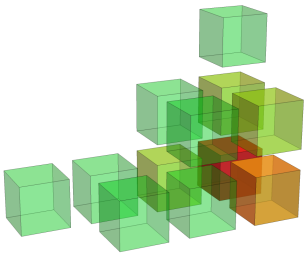


Figure 8: A partition of size 20, The height is given by the colour: Red is height 4, Orange is height 3, yellow is height 2 and green is height 1.

6 Concluding remarks

The concept of partitions has been around since the time of Euler, as a way of splitting up numbers into sums. MacMahon improved on it by finding a generating function for planar partitions and conjecturing such a formula for solid partitions. This formula turned out to be wrong. It is unclear if this was known to MacMahon.
As shown by [Knu70]'s article, the problem of enumerating solid partitions is not an easy one. The type of generating function that was found by Euler for the line case, and MacMahon for the planar case, is a very hard problem, as shown by the $e(n)$ series in Knuth's. Moreover simply calculating the amount of solid partitions of a certain size is complicated. Cutting edge at the moment of writing this thesis was the amount of solid partitions of 72. Knuths article shows a table to $n = 28$.
Instead of looking for a generating function, instead I looked at a weighted generating function conjectured by [CK18], see also [NP19] for a more general conjecture. This was a purely combinatorial conjecture, which follows from a more complicated geometrical conjecture. Using first Wolfram Mathematica and later C++ I checked a specialised version of this conjecture to order 25. This gives much new evidence for this, and by extension also the geometrical conjecture. Most of these programs can be run for longer and on more powerful computer systems to calculate even higher order terms of the conjecture. However the returns from this will be minimal, since it provides little more evidence than is already given. A more interesting route might be comparing the classical- and DT-weights to each other to see if pairs (or triplets or more) form of partitions with the same weighted sum under either weight.

A Python Scripts

A.1 Generating partitions

```
import sys
import time
import numpy as np
start = time.time()

sys.setrecursionlimit(5000)

print('starting')

dimension=3
globalSize=10 #the maximum size for which to generate partitions


def nextElement(theElement):
        currentElement = list(theElement)
        for i in range(dimension):
                if currentElement[i] + 1 < globalSize:
                        if i > 0:
                                for j in range(i):
                                        currentElement[j] = 0
                        currentElement[i] = currentElement[i]+1
                        return(currentElement)
        return 0


def partitionRecursion(theSize, x, y, z,theBaseSet,thePartitionSize):
        theAmount = 0
        thePartitionList = []

        if theSize == thePartitionSize:
                thePartitionList = [theBaseSet.copy()]
                theAmount = 1
#               print(theBaseSet, theSize)

        else:
                x2 = x + 1
                y2 = y
                z2 = z
                if thePartitionSize == 1:
                        pass
                elif x2 == thePartitionSize:
                        x2 = 0
                        y2 = y + 1
                        if y2 == thePartitionSize:
                                y2 = 0
                                z2 = z + 1
                                if z2 == thePartitionSize:
                                        return [theAmount,thePartitionList]
                if thePartitionSize > theSize:
                        loopLength = thePartitionSize-theSize+1
```

```
                        if x > 0:
                                loopLength = min(loopLength,theBaseSet[x-1][y][z]+1)
                        if y > 0:
                                loopLength = min(loopLength,theBaseSet[x][y-1][z]+1)
                        if z > 0:
                                loopLength = min(loopLength,theBaseSet[x][y][z-1]+1)

                        if loopLength < 2:
                                x2 = 0
                                y2 = y + 1
                                if y2 == thePartitionSize:
                                        y2 = 0
                                        z2 = z + 1
                                        if z2 == thePartitionSize:
                                                return [theAmount,thePartitionList]
                                if x == 0:
                                        y2 = 0
                                        z2 = z + 1
                                        if z2 == thePartitionSize:
                                                return [theAmount,thePartitionList]
                                if y == 0 and x ==0:
                                        return [theAmount,thePartitionList]

                        for i in range(loopLength):
                                newSize = theSize+i
                                theBaseSet[x][y][z] = i
                                theNextLevel = partitionRecursion(newSize,x2,y2,z2,theBaseS
                                theBaseSet[x][y][z] = 0
                                theAmount = theAmount + theNextLevel[0]
                                thePartitionList = thePartitionList + theNextLevel[1]
        return [theAmount,thePartitionList]

def partitionNumber(thePartitionSize, theDimension):
#       partitionList=[]
#       partitionAmount = 0

        theShape = tuple([thePartitionSize for i in range(theDimension)])
        theBaseSet =np.zeros(theShape,dtype=int)

        return partitionRecursion(0, 0, 0, 0, theBaseSet,thePartitionSize)

print("startingRecursion")
for i in range(globalSize+1):
        globSet = partitionNumber(i,dimension)
        print(str(i)+": "+str(globSet[0]))
        saveSet = np.array(globSet[1])
        saveSet.astype('float32').tofile('size'+str(i)+'dim'+str(saveSet.shape)+'.dat')

#print(globSet[1])
end = time.time()
print("Calculation time: " + str(end - start))
```

A.2 Calculating Classical Weights

A.2.1 Generating Planar Partitions

```
import sys
import time
import numpy as np
start = time.time()

sys.setrecursionlimit(5000)

print('starting')

dimension=3
globalSize=15


def partitionRecursion(theSize, x, y, z,theBaseSet):
        theAmount = 0
        thePartitionList = []
        if theSize == globalSize:
#               thePartitionList = [theBaseSet.copy()]
                theAmount = 1
#               print(theBaseSet, theSize)

        else:
                x2 = x + 1
                y2 = y
                z2 = z
                if x2 == globalSize:
                        x2 = 0
                        y2 = y + 1
                        if y2 == globalSize:
                                y2 = 0
                                z2 = z + 1
                                if z2 == globalSize:
                                        return [theAmount,thePartitionList]
                if globalSize > theSize:
                        loopLength = min(globalSize-theSize+1,2)

                        if x > 0:
                                loopLength = min(loopLength,theBaseSet[x-1][y][z]+1)
                        if y > 0:
                                loopLength = min(loopLength,theBaseSet[x][y-1][z]+1)
                        if z > 0:
                                loopLength = min(loopLength,theBaseSet[x][y][z-1]+1)

                        if loopLength < 2:
                                x2 = 0
                                y2 = y + 1
                                if y2 == globalSize:
                                        y2 = 0
                                        z2 = z + 1
                                        if z2 == globalSize:
                                                return [theAmount,thePartitionList]
```

```
                                    if x == 0:
                                            y2 = 0
                                            z2 = z + 1
                                            if z2 == globalSize:
                                                    return [theAmount, thePartitionList]
                                    if y == 0 and x ==0:
                                            return [theAmount, thePartitionList]

                            for i in range(loopLength):
                                    newSize = theSize+i
                                    theBaseSet[x][y][z] = i
                                    theNextLevel = partitionRecursion(newSize,x2,y2,z2,theBaseS
                                    if i > 0:
                                            thePartitionList.append(theBaseSet.copy())
#                                           print(x,y,z)
#                                           print(theBaseSet)
                                    theAmount = theAmount + theNextLevel[0]
                                    thePartitionList = thePartitionList + theNextLevel[1]
                                    theBaseSet[x][y][z] = 0
        return [theAmount, thePartitionList]

def partitionNumber(thePartitionSize, theDimension):
#       partitionList=[]
#       partitionAmount = 0

        theShape = tuple([thePartitionSize for i in range(theDimension)])
        theBaseSet =np.zeros(theShape,dtype=int)

        return partitionRecursion(0, 0, 0, 0, theBaseSet)

print("startingRecursion")
globSet = partitionNumber(globalSize,dimension)
print(globSet[0])
#print(globSet[1])
np.save('upto' + str(globalSize), np.array(globSet[1]))
end = time.time()
print(end - start)
```

A.2.2 Calculating the classical weight of a partition

```
import sys
import time
import numpy as np
import math
start = time.time()

sys.setrecursionlimit(5000)

print('starting')
start = time.time()

dimension=3
globalSize=15

globShape = tuple([globalSize for i in range(dimension)])
globalPartition = np.zeros(globShape,dtype=int)

globalPartition[0,0,0]=3
globalPartition[1,0,0]=2
globalPartition[0,1,0]=1
globalPartition[0,0,1]=1
globalPartition[1,1,0]=1


planarSetAll = np.load('upto'+str(globalSize)+'.npy')
planarLSet = []
for xi in planarSetAll:
        if (xi > globalPartition).any() == False:
                planarLSet.append(xi)
planarSet = np.array(planarLSet)

def weightRecursion(theSum, theWeight,newWeight,theDepth):
        theWeight = 0.0
        for i in range(theDepth):
                for j in range(1,globalPartition[0,0,0] - theSum[0,0,0] + 1):
                        newSum = theSum + j*planarSet[i]
                        newnewWeight = newWeight/(math.factorial(j))
                        if (newSum == globalPartition).all():
                                return(newnewWeight)
                        elif (newSum[0,0,0] > globalPartition[0,0,0]).all():
                                return(0)
                        else:
                                theWeight = theWeight + weightRecursion(newSum, theWeight,r
        return theWeight

#print(planarSet.shape[0])
print(weightRecursion(np.zeros(globShape,dtype=int),0.0,1.0,planarSet.shape[0]))
end = time.time()
print(end - start)
```

B Calculating0 the polynomial weight

B.1 Cell 1: Importing

```
pSize = 7;
pAmount = 307;
file = "size" <> ToString[pSize] <> "dim(" <> ToString[pAmount] <>
  ",␣" <> ToString[pSize] <> ",␣" <> ToString[pSize] <> ",␣" <>
  ToString[pSize] <> ").dat"
data = BinaryReadList["C:/Users/Frank/Desktop/scriptie/" <> file ,
   "Real32"];
partitionList = ArrayReshape[data, {pAmount, pSize, pSize, pSize}];
pPoly[partition_] := Module[{THEpolynomial = 0, THEfunction},
  For[i1 = 0, i1 < pSize, i1++,
   For[i2 = 0, i2 < pSize, i2++,
    For[i3 = 0, i3 < pSize, i3++,
     For[i4 = 0, i4 < partition[[i1 + 1, i2 + 1, i3 + 1]], i4++,
      THEpolynomial = THEpolynomial + t1^i1 t2^i2 t3^i3 t4^i4;
      ]
     ]
    ]
   ];
  THEpolynomial
  ]
pPolyList = Map[pPoly, partitionList]
```

B.2 Cell 2: Expressions to functions

```
exprToFunc[expr_] := Module[{func},
  func = Function[
    Evaluate[
     expr /. Thread[{t1, t2, t3, t4} ->
        Array[Slot, Length[{t1, t2, t3, t4}]]]]
    ];
  func
  ]
exprToFunc2[expr_] := Module[{func},
  func = Function[
    Evaluate[
     expr /. Thread[{d, l1, l2, l3} ->
        Array[Slot, Length[{d, l1, l2, l3}]]]]
    ];
  func
  ]
pFuncList = Map[exprToFunc, pPolyList]
```

B.3 Cell 3: Defining the weight function

```
UnspecialisedWeight[partition_] :=
 Module[{Zpi, Zbar, Vpi, sumList, listDim, sumList2, listDim2, wpi2,
   wpi2F, Lpi, LpiF},
  Zpi = partition;
  Zbar[t1_, t2_, t3_, t4_] := Zpi[t1^-1, t2^-1, t3^-1, t4^-1];
  Vpi[t1_, t2_, t3_, t4_] :=
   Zpi[t1, t2, t3, t4] + Zbar[t1, t2, t3, t4]/(t1 t2 t3 t4) - (
    Zpi[t1, t2, t3, t4] Zbar[t1, t2, t3,
      t4] (1 - t1) (1 - t2) (1 - t3) (1 - t4))/(t1 t2 t3 t4);
  ShiftNumber = pSize + 1;
  sumList =
   CoefficientList[
    Vpi[t1, t2, t3, t4]*(t1 t2 t3 t4)^ShiftNumber, {t1, t2, t3, t4}];
  listDim = Dimensions[sumList];

  wpi2F[l1_, l2_, l3_, l4_] :=
   Product[If[Sign[sumList[[a1, a2, a3, a4]]] == 0,
     1, ((a1 - ShiftNumber) l1 + (a2 - ShiftNumber) l2 + (a3 -
           ShiftNumber) l3 + (a4 - ShiftNumber) l4)^-sumList[[a1, a2,
       a3, a4]]], {a1, 1, listDim[[1]]}, {a2, 1, listDim[[2]]}, {a3,
     1, listDim[[3]]}, {a4, 1, listDim[[4]]}];

  sumList2 = CoefficientList[Zpi[t1, t2, t3, t4], {t1, t2, t3, t4}];
  listDim2 = Dimensions[sumList2];

  LpiF[d1_, d2_, d3_, d4_, l1_, l2_, l3_, l4_] :=
   Product[If[Sign[sumList2[[a1, a2, a3, a4]]] == 0,
     1, ((a1 + d1) l1 + (a2 + d2) l2 + (a3 + d3) l3 + (a4 + d4) l4)^
      sumList2[[a1, a2, a3, a4]]], {a1, 1, listDim2[[1]]}, {a2, 1,
     listDim2[[2]]}, {a3, 1, listDim2[[3]]}, {a4, 1, listDim2[[4]]}];
  Simplify[
   wpi2F[l1, l2, l3, -l1 - l2 - l3]*
    LpiF[0, 0, 0, -d, l1, l2, l3, -l1 - l2 - l3]^2]
  ]
Weight[partition_] := Module[{exprF, returnExpr},
  exprF = exprToFunc2[UnspecialisedWeight[partition]];
  returnExpr =
   Simplify[Sqrt[(-1)^pSize exprF[d, l1, l2, -l1 - l2]], d > 100];
  Print[partition[t1, t2, t3, t4], ", ", returnExpr];
  returnExpr
  ]
```

B.4 Cell 4: Calculating the weights and adding

```
WeightSet = ParallelMap[Weight, pFuncList]
Expand[Simplify[Total[WeightSet]]]
```

B.5 Cell 5: Calculation the expected value

```
M[q_] := Sum[{1, 1, 3, 6, 13, 24, 48, 86, 160, 282, 500, 859, 1479,
     2485, 4167, 6879, 11297, 18334, 29601, 47330, 75278, 118794,
     186475, 290783, 451194, 696033, 1068745}[[i]] q^(i - 1), {i, 21}]
coeffList =
  Map[Expand, CoefficientList[Series[M[q]^d, {q, 0, 20}], q]];
TableForm[coeffList]
```

C Calculation Results

The program outputs the coefficients $a1$ to $a25$, and as an expression in a variable d, fi for order i, in mathematica input format. Here fi is the sum of all the weights of partitions of size i.

C.1 Source returned

```
f1 = 1*d^1;

f2 = 5/2*d^1 + 1/2*d^2;

f3 = 10/3*d^1 + 5/2*d^2 + 1/6*d^3;

f4 = 21/4*d^1 + 155/24*d^2 + 5/4*d^3 + 1/24*d^4;

f5 = 26/5*d^1 + 163/12*d^2 + 115/24*d^3 + 5/12*d^4 + 1/120*d^5;

f6 = 25/3*d^1 + 8597/360*d^2 + 217/16*d^3 + 305/144*d^4 + 5/48*d^5 + 1/720*d^6;

f7 = 50/7*d^1 + 233/6*d^2 + 11411/360*d^3 + 367/48*d^4 + 95/144*d^5 +
   1/48*d^6 + 1/5040*d^7;

f8 = 85/8*d^1 + 39709/672*d^2 + 18709/288*d^3 + 128167/5760*d^4 +
   419/144*d^5 + 91/576*d^6 + 1/288*d^7 + 1/40320*d^8;

f9 = 91/9*d^1 + 210571/2520*d^2 + 2207983/18144*d^3 + 5339/96*d^4 +
   175669/17280*d^5 + 33/40*d^6 + 53/1728*d^7 + 1/2016*d^8 +
   1/362880*d^9;

f10 = 13*d^1 + 2967379/25200*d^2 + 4258643/20160*d^3 +
   1128287/9072*d^4 + 69287/2304*d^5 + 582613/172800*d^6 +
   1067/5760*d^7 + 121/24192*d^8 + 1/16128*d^9 + 1/3628800*d^10;

f11 = 122/11*d^1 + 5563/36*d^2 + 2189501/6300*d^3 +
   46248031/181440*d^4 + 2848621/36288*d^5 + 79789/6912*d^6 +
   150023/172800*d^7 + 593/17280*d^8 + 17/24192*d^9 + 1/145152*d^10 +
   1/39916800*d^11;

f12 = 35/2*d^1 + 13634311/66528*d^2 + 65964713/120960*d^3 +
   5305417589/10886400*d^4 + 269648441/1451520*d^5 +
   60151717/1741824*d^6 + 233743/69120*d^7 + 2635951/14515200*d^8 +
   871/161280*d^9 + 151/1741824*d^10 + 1/1451520*d^11 +
   1/479001600*d^12;

f13 = 170/13*d^1 + 118465/462*d^2 + 54809941/66528*d^3 +
   15213979/17280*d^4 + 4421016113/10886400*d^5 +
   134772349/1451520*d^6 + 99492497/8709120*d^7 + 384071/483840*d^8 +
   462547/14515200*d^9 + 17/23040*d^10 + 83/8709120*d^11 +
   1/15966720*d^12 + 1/6227020800*d^13;

f14 = 125/7*d^1 + 829092461/2522520*d^2 + 4011517997/3326400*d^3 +
   2421318793/1596672*d^4 + 3621151109/4354560*d^5 +
   9965349133/43545600*d^6 + 428582939/12441600*d^7 +
   364785083/121927680*d^8 + 897511/5806080*d^9 +
```

```
   15499/3225600*d^10 + 7751/87091200*d^11 + 181/191600640*d^12 +
   1/191600640*d^13 + 1/87178291200*d^14;

f15 = 52/3*d^1 + 30491257/77220*d^2 + 121729348007/70945875*d^3 +
   18781309087/7484400*d^4 + 23158476851/14370048*d^5 +
   761623669/1451520*d^6 + 8830763927/93312000*d^7 +
   871821233/87091200*d^8 + 235406767/365783040*d^9 +
   446401/17418240*d^10 + 828497/1306368000*d^11 +
   3067/319334400*d^12 + 7/82114560*d^13 + 1/2490808320*d^14 +
   1/1307674368000*d^15;

f16=(341 d)/16 + (2837414269 d^2)/5765760 + (
 578373381637 d^3)/242161920 + (64741946507729 d^4)/16144128000 + (
 713696497517 d^5)/239500800 + (1300958306899 d^6)/1149603840 + (
 8378430673 d^7)/34836480 + (634648928233 d^8)/20901888000 + (
 721855979 d^9)/304819200 + (171009001 d^10)/1463132160 + (
 64279 d^11)/17418240 + (8547401 d^12)/114960384000 + (
 449 d^13)/479001600 + (
 211 d^14)/29889699840 + d^15/34871316480 + d^16/20922789888000

 f17=(290 d)/17 + (83104123 d^2)/144144 + (393352882789 d^3)/121080960 + (
 7531038138277 d^4)/1210809600 + (
 2306235614197979 d^5)/435891456000 + (
 1663948791367 d^6)/718502400 + (657609112739 d^7)/1149603840 + (
 44312473243 d^8)/522547200 + (1151014025183 d^9)/146313216000 + (
 858501653 d^10)/1828915200 + (26781989 d^11)/1463132160 + (
 673753 d^12)/1437004800 + (900673 d^13)/114960384000 + (
 3119 d^14)/37362124800 + (
 113 d^15)/209227898880 + d^16/523069747200 + d^17/355687428096000

 f18=(455 d)/18 + (2389178593 d^2)/3392928 + (
 150524451691 d^3)/34594560 + (614238791240993 d^4)/65383718400 + (
 4742448362041849 d^5)/523069747200 + (
 106581944352007697 d^6)/23538138624000 + (
 4924140366781 d^7)/3832012800 + (761203644347 d^8)/3448811520 + (
 22267357057 d^9)/928972800 + (1488913250981 d^10)/877879296000 + (
 1166338601 d^11)/14631321600 + (3641421571 d^12)/1448500838400 + (
 815987 d^13)/15328051200 + (516851 d^14)/689762304000 + (
 227 d^15)/33210777600 + (
 241 d^16)/6276836966400 + d^17/8369115955200 + d^18/6402373705728000

 f19=(362 d)/19 + (987632677 d^2)/1225224 + (
 88258083296987 d^3)/15437822400 + (
 251762257859633 d^4)/18162144000 + (
 1969218621137941 d^5)/130767436800 + (
 4449453116647657 d^6)/523069747200 + (
 64644571430999753 d^7)/23538138624000 + (
 31065415075943 d^8)/57480192000 + (6549091674977 d^9)/96566722560 + (
 327890593423 d^10)/58525286400 + (273381251639 d^11)/877879296000 + (
 1055487847 d^12)/89413632000 + (444553891 d^13)/1448500838400 + (
 3263189 d^14)/597793996800 + (4116419 d^15)/62768369664000 + (
 5417 d^16)/10461394944000 + d^17/392302310400 + d^18/142274971238400 \
+ d^19/121645100408832000
```

```
f20=(273 d)/10 + (899444162587 d^2)/931170240 + (
 917620047693487 d^3)/123502579200 + (
 30925601998949741 d^4)/1543782240000 + (
 18177971818398547 d^5)/747242496000 + (
 32273304273038993 d^6)/2092278988800 + (
 1923437693642509 d^7)/342372925440 + (
 1181215319351541769 d^8)/941525544960000 + (
 3474945664739683 d^9)/19313344512000 + (
 26471187948451 d^10)/1545067560960 + (
 11127861979 d^11)/10032906240 + (
 9579475317073 d^12)/193133445120000 + (
 44794004273 d^13)/28970016768000 + (
 5073670957 d^14)/150644087193600 + (12803951 d^15)/25107347865600 + (
 6639617 d^16)/1255367393280000 + (17 d^17)/465813504000 + (
 271 d^18)/1707299654860800 + d^19/2560949482291200 + \
d^20/2432902008176640000

f21 = 500/21*d^1 + 533663270381/488864376*d^2 +
   14618295302019821/1539922784400*d^3 +
   10527050039758121/370507737600*d^4 +
   491656876451578631/12825267840000*d^5 +
   425295477171580931/15692092416000*d^6 +
   48033103277735383/4345502515200*d^7 +
   4755823673333657/1711864627200*d^8 +
   810871665330984389/1797457858560000*d^9 +
   946079161734919/19313344512000*d^10 +
   84532110948181/23176013414400*d^11 +
   732200783393/3862668902400*d^12 +
   12090795900037/1738201006080000*d^13 +
   9752245211/53801459712000*d^14 +
   10608943489/3163525831065600*d^15 + 1097969/25107347865600*d^16 +
   114547/289700167680000*d^17 + 311/129340882944000*d^18 +
   1/107452425830400*d^19 + 1/48658040163532800*d^20 +
   1/510909421717094400000*d^21;

f22 = 305/11*d^1 + 50794984507/39395664*d^2 +
   13079651013941/1086365280*d^3 +
   3905500199657297849/98555058201600*d^4 +
   145916106970774739/2470051584000*d^5 +
   9499473039043044923/205204285440000*d^6 +
   146492495687852287/6974263296000*d^7 +
   9521414957690123/1614043791360*d^8 +
   405190983395861659/376610217984000*d^9 +
   208811944078741921/158176291553280000*d^10 +
   78565119808177/7023034368000*d^11 +
   340267381949287/509872295116800*d^12 +
   2194248820513/77253378048000*d^13 +
   61689387323/70946979840000*d^14 +
   28932580441/1506440871936000*d^15 +
   1932115697/6327051662131200*d^16 + 1739537/502146957312000*d^17 +
   135647/4924902850560000*d^18 + 2539/170729965486080000*d^19 +
   43/83413783137484800*d^20 + 1/973160803270656000*d^21 +
   1/11240007277776076800000*d^22;
```

```
f23 = 530/23*d^1 + 855318055/596904*d^2 +
   9705006597945563/645300976320*d^3 +
   13409225827019312 87/24638764550400*d^4 +
   65941970251547414309/739162936512000*d^5 +
   10280877316774478761/133382785536000*d^6 +
   309595139382071817899/8002967132160000*d^7 +
   6814887454767800623/564915326976000*d^8 +
   346554951152750099/141228831744000*d^9 +
   667404404552187301/1977203644416000*d^10 +
   5103291765400847947/15817629155328000 0*d^11 +
   506752884911149/231760134144000*d^12 +
   18122495293379/169957431705600*d^13 +
   2627572639957/695280402432000*d^14 +
   4423646325577/45193226158080000*d^15 +
   58671870889/31635258310656000*d^16 +
   809984699/31635258310656000*d^17 +
   6528107/25609494822912000*d^18 + 114977/64023737057280000*d^19 +
   25301/2919482409811968000*d^20 + 79/2919482409811968000*d^21 +
   1/2043637686868377 6000*d^22 + 1/2585201673888497664 0000*d^23;

f24 = 425/12*d^1 + 215706423518371/128501493120*d^2 +
   57895588197398841/30974446863360*d^3 +
   4795363294573495257157/65046338413056000*d^4 +
   4266228805736636082481/32254382684160000*d^5 +
   1067781876192312516791 03/851515702861824000*d^6 +
   1330255882707558383257/19207121117184000*d^7 +
   2748148701317025355279 9/1152427267031040000*d^8 +
   13499099121212203841/2510734786560000*d^9 +
   3126066133870177456 87/379623099727872000*d^10 +
   11147179775427164797/126541033242624000*d^11 +
   5104190021622538859 9/7592461994557440000*d^12 +
   14218906763055121/38240422133760000*d^13 +
   3597170294524997/2386202341146624 00*d^14 +
   247212479057/547796680704000*d^15 +
   75998469706169/7592461994557440000*d^16 +
   13001317579/79088145776640000*d^17 +
   6415628143/32267963476869120 00*d^18 +
   5380477/307313937874944000*d^19 +
   38550301/3503378891774361600 00*d^20 +
   2323/4865804016353280000*d^21 + 331/24523652242420531200 0*d^22 +
   1/4496002911110430720 00*d^23 + 1/6204484017333943936 0000*d^24;


f25 = 651/25*d^1 + 263518478777/143161200*d^2 +
   106272164372674519/4638100767300*d^3 +
   76334112054515579321/774361171584000*d^4 +
   15679403999282992412 99227/8130792301632000000*d^5 +
   70870048870835848378991/354798209525760000*d^6 +
   10289954672967168872 5999/851515702861824000*d^7 +
   87796182968917523490 1/19207121117184000*d^8 +
   65462299968536837225939/57621633351552000 00*d^9 +
   3347098711929931873/1738201006080000*d^10 +
   79031193281515801 01/34511190884352000*d^11 +
   2471356989198268937/126541033242624000*d^12 +
```

```
507169852822130478737/417585409700659200000*d^13 +
6930208772314567/124281371934720000*d^14 +
454964757402761/238620234114662400*d^15 +
882252424139/18077290463232000*d^16 +
35563825311733/3796230997278720000*d^17 +
2128656913/158176291553280000*d^18 +
463780249/32267963476869120000*d^19 +
6581791/5838964819623936000*d^20 +
1012579/1592444950806528000000*d^21 + 1/40102780354560000*d^22 +
173/2697601746666258432000*d^23 + 1/10340806695553990656000*d^24 +
1/1551121004333098598400000000*d^25;
```

C.2 Table

This is the same data compiled in a more readable table.

0	d	d^2	d^3	d^4	d^5	d^6
q	1	0	0	0	0	0
q^2	$\frac{5}{2}$	$\frac{1}{2}$	0	0	0	0
q^3	$\frac{10}{3}$	$\frac{5}{2}$	$\frac{1}{6}$	0	0	0
q^4	$\frac{21}{4}$	$\frac{155}{24}$	$\frac{5}{4}$	$\frac{1}{24}$	0	0
q^5	$\frac{26}{5}$	$\frac{163}{12}$	$\frac{115}{24}$	$\frac{5}{12}$	$\frac{1}{120}$	0
q^6	$\frac{25}{3}$	$\frac{8597}{360}$	$\frac{217}{16}$	$\frac{305}{144}$	$\frac{5}{48}$	$\frac{1}{720}$
q^7	$\frac{50}{7}$	$\frac{233}{6}$	$\frac{11411}{360}$	$\frac{367}{48}$	$\frac{95}{144}$	$\frac{1}{48}$
q^8	$\frac{85}{8}$	$\frac{39709}{672}$	$\frac{18709}{288}$	$\frac{128167}{5760}$	$\frac{419}{144}$	$\frac{91}{576}$
q^9	$\frac{91}{9}$	$\frac{210571}{2520}$	$\frac{2207983}{18144}$	$\frac{5339}{96}$	$\frac{175669}{17280}$	$\frac{33}{40}$
q^{10}	13	$\frac{2967379}{25200}$	$\frac{4258643}{20160}$	$\frac{1128287}{9072}$	$\frac{69287}{2304}$	$\frac{582613}{172800}$
q^{11}	$\frac{122}{11}$	$\frac{5563}{36}$	$\frac{2189501}{6300}$	$\frac{46248031}{181440}$	$\frac{2848621}{36288}$	$\frac{79789}{6912}$
q^{12}	$\frac{35}{2}$	$\frac{13634311}{66528}$	$\frac{65964713}{120960}$	$\frac{5305417589}{10886400}$	$\frac{269648441}{1451520}$	$\frac{60151717}{1741824}$
q^{13}	$\frac{170}{13}$	$\frac{118465}{462}$	$\frac{54809941}{66528}$	$\frac{15213979}{17280}$	$\frac{4421016113}{10886400}$	$\frac{134772349}{1451520}$
q^{14}	$\frac{125}{7}$	$\frac{829092461}{2522520}$	$\frac{4011517997}{3326400}$	$\frac{2421318793}{1596672}$	$\frac{3621151109}{4354560}$	$\frac{9965349133}{43545600}$
q^{15}	$\frac{52}{3}$	$\frac{30491257}{77220}$	$\frac{121729348007}{70945875}$	$\frac{18781309087}{7484400}$	$\frac{23158476851}{14370048}$	$\frac{761623669}{1451520}$
q^{16}	$\frac{341}{16}$	$\frac{2837414269}{5765760}$	$\frac{578373381637}{242161920}$	$\frac{64741946507729}{16144128000}$	$\frac{713696497517}{239500800}$	$\frac{1300958306899}{1149603840}$
q^{17}	$\frac{290}{17}$	$\frac{83104123}{144144}$	$\frac{393352882789}{121080960}$	$\frac{7531038138277}{1210809600}$	$\frac{2306235614197979}{435891456000}$	$\frac{1663948791367}{718502400}$
q^{18}	$\frac{455}{18}$	$\frac{2389178593}{3392928}$	$\frac{150524451691}{34594560}$	$\frac{614238791240993}{65383718400}$	$\frac{47424483620418 49}{523069747200}$	$\frac{1065819443520076 97}{235381138624000}$
q^{19}	$\frac{362}{19}$	$\frac{987632677}{1225224}$	$\frac{88258083296987}{15437822400}$	$\frac{251762257859633}{18162144000}$	$\frac{1969218621137941}{130767436800}$	$\frac{4449453116647657}{523069747200}$
q^{20}	$\frac{273}{10}$	$\frac{899444162587}{931170240}$	$\frac{917620047693487}{123502579200}$	$\frac{309256019989497 41}{1543782240000}$	$\frac{181779718183985 47}{747242496000}$	$\frac{3227330427303899 3}{2092278988800}$
q^{21}	$\frac{500}{21}$	$\frac{533663270381}{488864376}$	$\frac{146182953020198 21}{1539922784400}$	$\frac{105270500397581 21}{370507737600}$	$\frac{491656876451578631}{1282526784000 0}$	$\frac{4252954771715809 31}{1569209241600 0}$
q^{22}	$\frac{305}{11}$	$\frac{50794984507}{39395664}$	$\frac{13079651013941}{1086365280}$	$\frac{390550019965729 7849}{9855505820160 0}$	$\frac{145916106970774739}{247005158400 0}$	$\frac{9499473039043044923}{2052042854400 00}$
q^{23}	$\frac{530}{23}$	$\frac{855318055}{596904}$	$\frac{970500659794 5563}{645300976320}$	$\frac{134092258270193 1287}{2463876455040 0}$	$\frac{659419702515474 14309}{739162936512000}$	$\frac{1028087731677447 8761}{1333827855360 00}$
q^{24}	$\frac{425}{12}$	$\frac{215706423518371}{128501493120}$	$\frac{578955888197398841}{3097444863360}$	$\frac{479536329457349525 7157}{6504633841305600 0}$	$\frac{426622880573663 6082481}{3225438268416000 0}$	$\frac{1067781876192312 51679103}{8515157028618240 00}$
q^{25}	$\frac{651}{25}$	$\frac{263518478777}{143161200}$	$\frac{106272164372674519}{4638100767300}$	$\frac{763341120545155793 21}{774361171584000}$	$\frac{156794039992829924 1299227}{8130792301632000000}$	$\frac{708700488708358483 78991}{354798209525760000}$

d^7	d^8	d^9	d^{10}	d^{11}
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
$\frac{1}{5040}$	0	0	0	0
$\frac{1}{288}$	$\frac{1}{40320}$	0	0	0
$\frac{53}{1728}$	$\frac{1}{2016}$	$\frac{1}{362880}$	0	0
$\frac{1067}{5760}$	$\frac{121}{24192}$	$\frac{1}{16128}$	$\frac{1}{3628800}$	0
$\frac{150023}{172800}$	$\frac{593}{17280}$	$\frac{17}{24192}$	$\frac{1}{145152}$	$\frac{1}{39916800}$
$\frac{233743}{69120}$	$\frac{2635951}{14515200}$	$\frac{871}{161280}$	$\frac{151}{1741824}$	$\frac{1}{1451520}$
$\frac{99492497}{8709120}$	$\frac{384071}{483840}$	$\frac{462547}{14515200}$	$\frac{17}{23040}$	$\frac{83}{8709120}$
$\frac{428582939}{12441600}$	$\frac{364785083}{121927680}$	$\frac{897511}{5806080}$	$\frac{15499}{3225600}$	$\frac{7751}{87091200}$
$\frac{8830763927}{93312000}$	$\frac{87821233}{87091200}$	$\frac{235406767}{365783040}$	$\frac{446401}{17418240}$	$\frac{828497}{1306368000}$
$\frac{8378430673}{34836480}$	$\frac{634648928233}{20901888000}$	$\frac{721855979}{304819200}$	$\frac{171009001}{1463132160}$	$\frac{64279}{17418240}$
$\frac{657609112739}{1149603840}$	$\frac{44312473243}{522547200}$	$\frac{1151014025183}{1463132160000}$	$\frac{858501653}{18289152000}$	$\frac{26781989}{1463132160}$
$\frac{4924140366781}{3832012800}$	$\frac{761203644347}{3448811520}$	$\frac{22267357057}{928972800}$	$\frac{1488913250981}{877879296000}$	$\frac{1166338601}{146313216000}$
$\frac{64644571430999753}{23538138624000}$	$\frac{31065415075943}{5748019200}$	$\frac{6549091674977}{96566722560}$	$\frac{327890593423}{58525286400}$	$\frac{273381251639}{877879296000}$
$\frac{1923437693642509}{342372925440}$	$\frac{1181215319351541769}{9415255449600000}$	$\frac{3474945664739683}{19313344512000}$	$\frac{264711879484 51}{1545067560960}$	$\frac{11127861979}{10032906240}$
$\frac{4803310327735383}{4345502515200}$	$\frac{4755823673333657}{1711864627200}$	$\frac{8108716653309843 89}{17974578585600 00}$	$\frac{946079161734919}{19313344512000}$	$\frac{84532110948181}{23176013414400}$
$\frac{1464924956878522 87}{6974263296000}$	$\frac{952141495769012 3}{1614043791360}$	$\frac{4051909833958616 59}{3766102179840 00}$	$\frac{2088119440787419 2101}{1581762915532800 00}$	$\frac{7856511980817 7}{7023034368000}$
$\frac{3095951393820718 17899}{8002967132160000}$	$\frac{6814887454767800 623}{5649153269760 00}$	$\frac{3465549511527500 99}{1412288317440 00}$	$\frac{6674044045521873 01}{1977203644416000}$	$\frac{5103291765400847 947}{1581762915532800 00}$
$\frac{1330255882707558383257}{19207121117184000}$	$\frac{2748148701317025355279 9}{1152427267031040000}$	$\frac{1349900991212122038 41}{2510734786560000}$	$\frac{3126066133870177 45687}{379623099727872000}$	$\frac{1114717977542716 4797}{1265410332426240 00}$
$\frac{1028995467296716887259 99}{851515702861824000}$	$\frac{8779618296891752349 01}{19207121117184000}$	$\frac{6546229996853683722 5939}{5762136335155200000}$	$\frac{3347098711929931873}{1738201006080000}$	$\frac{7903119328151580101}{345111908843520 00}$

d^{12}	d^{13}	d^{14}	d^{15}	d^{16}
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
$\frac{1}{479001600}$	0	0	0	0
$\frac{1}{15966720}$	$\frac{1}{6227020800}$	0	0	0
$\frac{181}{191600640}$	$\frac{1}{191600640}$	$\frac{1}{87178291200}$	0	0
$\frac{3067}{319334400}$	$\frac{7}{82114560}$	$\frac{1}{2490808320}$	$\frac{1}{1307674368000}$	0
$\frac{8547401}{1149603840 00}$	$\frac{449}{479001600}$	$\frac{211}{29889699840}$	$\frac{1}{34871316480}$	$\frac{1}{20922789888000}$
$\frac{673753}{1437004800}$	$\frac{900673}{1149603840 00}$	$\frac{3119}{37362124800}$	$\frac{113}{209227898880}$	$\frac{1}{523069747200}$
$\frac{3641421571}{1448500838400}$	$\frac{815987}{153280512 00}$	$\frac{516851}{689762304000}$	$\frac{227}{33210777600}$	$\frac{241}{6276836966400}$
$\frac{1055487847}{894136320 00}$	$\frac{444553891}{1448500838400}$	$\frac{3263189}{5977939968 00}$	$\frac{4116419}{627683696640 00}$	$\frac{5417}{1046139494400 0}$
$\frac{9579475317073}{193133445120000}$	$\frac{447940042 73}{289700167680 00}$	$\frac{5073670957}{1506440871936 00}$	$\frac{12803951}{251073478655 00}$	$\frac{6639617}{1255367393280000}$
$\frac{732200783393}{3862668902400}$	$\frac{1209079590 0037}{1738201006080000}$	$\frac{9752245211}{538014597120 00}$	$\frac{10608943489}{3163525831065600}$	$\frac{1097969}{251073478656 00}$
$\frac{340267381949287}{509872295116800}$	$\frac{2194248820513}{7725337804800 0}$	$\frac{61689387323}{709469798400 00}$	$\frac{28932580441}{1506440871936 000}$	$\frac{1932115697}{6327051662131200}$
$\frac{506752884911149}{2317601341440 00}$	$\frac{1818224952933 79}{1699574317056 00}$	$\frac{2627572639957}{6952804024320 00}$	$\frac{4423646325577}{4519322661580800 00}$	$\frac{5867187088 9}{3163525831065 6000}$
$\frac{5104190021622538859 9}{7592461994557440000}$	$\frac{1421890676305512 1}{3824042213376000 0}$	$\frac{3597170294524997}{2386202341146624 00}$	$\frac{247212479057}{5477966807040 00}$	$\frac{7599846970616 9}{7592461994557 440000}$
$\frac{2471356989198268937}{126541033242624000}$	$\frac{5071698528221304787 37}{4175854097006592000 0}$	$\frac{6930208772314567}{1242813719347200 00}$	$\frac{454964757402761}{2386202341146624 00}$	$\frac{882252424139}{18077290463232000}$

d^{17}	d^{18}	d^{19}	d^{20}
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
$\frac{1}{355687428096000}$	0	0	0
$\frac{1}{8369115955200}$	$\frac{1}{6402373705728000}$	0	0
$\frac{1}{392302310400}$	$\frac{1}{142274971238400}$	$\frac{1}{121645100408832000}$	0
$\frac{17}{465813504000}$	$\frac{271}{17072996548608000}$	$\frac{1}{2560949482291200}$	$\frac{1}{2432902008176640000}$
$\frac{114547}{28970016768000}$	$\frac{311}{129340882944000}$	$\frac{1}{107452425830400}$	$\frac{1}{48658040163532800}$
$\frac{1739537}{502146957312000}$	$\frac{135647}{4924902850560000}$	$\frac{2539}{170729965486080000}$	$\frac{43}{83413783137484800}$
$\frac{809984699}{31635258310656000}$	$\frac{6528107}{256094948229120000}$	$\frac{114977}{64023737057280000}$	$\frac{25301}{2919482409811968000}$
$\frac{13001317579}{790881457766400000}$	$\frac{6415628143}{3226796347686912000}$	$\frac{5380477}{3073139378749440000}$	$\frac{38550301}{350337889177436160000}$
$\frac{35563825311733}{37962309972787200000}$	$\frac{2128656913}{1581762915532800000}$	$\frac{463780249}{3226796347686912000}$	$\frac{6581791}{5838964819623936000}$

d^{21}	d^{22}	d^{23}	d^{24}	d^{25}
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
$\frac{1}{5109094217170944000}$	0	0	0	0
$\frac{1}{9731608032706560000}$	$\frac{1}{1124000727777607680000}$	0	0	0
$\frac{79}{2919482409811968000}$	$\frac{1}{204636768683776000}$	$\frac{1}{2585201673884976640000}$	0	0
$\frac{2323}{4865804016353280000}$	$\frac{331}{24523652242420531200}$	$\frac{1}{44960029111104307200}$	$\frac{1}{620448401733239439360000}$	0
$\frac{1012579}{159244495080652800000}$	$\frac{1}{40102780354560000}$	$\frac{173}{2697601746666258432000}$	$\frac{1}{10340806695553990656000}$	$\frac{1}{15511210043330985984000000}$

References

[CK18] Y. Cao and M. Kool. "Zero dimensional Donaldson-Thomas invariants of Calabi-Yau 4-folds". In: *Advances in mathematics* 338 (2018), pp. 601–648.

[Knu70] D. E. Knuth. "A note on solid partitions". In: *Mathematics of computation* 24.112 (1970). http://www.ams.org/journ 24-112/S0025-5718-1970-0277401-7/S0025-5718-1970-0277401-7.pdf.

[NP19] N. Nekrasov and N. Piazzalunga. "Magnificent Four with Colors". In: *Communications in Math and Theoretical Physics* (2019). https://arxiv.org/pdf/1808.05206.pdf.

[OEI] OEIS. *Online encyclopedia of integer sequences*. https://oeis.org/A000293.

[Ren19] J. Rennemo. Unpublished Manuscript. 2019.

[Sta] Richard P. Stanley. *Enumerative Combinatorics*. Vol. 2. Corollary 7.20.3.

[Wil] Herbert S Wilf. *Generatingfunctionology*. https://www.math.upenn.edu/ wilf/gfologyLinked2.pdf.