Uniform Random Samples for Second-Order Restricted k-Compositions

Thomas Stein
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands

ABSTRACT

This paper introduces a method for finding uniformly random second-order restricted k-compositions of N. This method involves a new algorithm which maps an integer to a composition, and uses a uniform random integer as the input for this mapping. The paper also examines existing methods to randomly select a composition, and discusses the benefits of the new algorithm over those methods.

Keywords

uniform random samples, composition, k-composition, mapping, bijection, second-order restricted, constraint problem, combinatorics

1. INTRODUCTION

This paper introduces a solution to the following problem: Given a positive integer N and a vector $R = \langle R_i \in [0, N] \mid 1 \leq i \leq k \rangle$ (i.e. a list of k positive integers less than or equal to N), find a random list $a = \langle a_i \in [0, R_i] \mid 1 \leq i \leq k \rangle$ such that $a_1 + ... + a_k = N$ (i.e. a list of k positive integers where each integer at index i is less than or equal to R_i, such that the sum of the chosen integers is equal to N). Random means that for all lists that satisfy these constraints, the probabilities that one is generated are equal.

These lists $\langle a_1, ..., a_k \rangle$ a are a specific subset of integer compositions, called second-order restricted by Page [3], or $(m_1, ..., m_n)$-bounded by Walsh [6]. This list $(m_1, ..., m_k)$ is analogous to the vector R defined above. There has been prior research on these compositions—this has mostly been related to finding the number of solutions [4], generating solutions iteratively in lexicographical order [6], and generating every solution [2]. Compositions are explained more in-depth in section 2.

2. BACKGROUND INFORMATION

A composition of N is a way to write an integer N as a sum of any amount of positive numbers. A k-composition of N is a way to write an integer N as a sum of k positive parts. Two compositions are considered distinct when they contain the same set of integers in different orders: $\langle 1, 2 \rangle$ and $\langle 2, 1 \rangle$ are two different 2-compositions of 3 [4, p.107]. A composition is called "weak" if its parts are allowed to be 0; The distinction is important to note because, for compositions where there is no k specified, allowing 0 in its parts results in infinite solutions: $\langle 5 \rangle$, $\langle 5, 0 \rangle$, $\langle 5, 0, 0 \rangle$, etc. are all valid weak compositions of 5.

Generating functions that calculate the amount of solutions for given N and k have been known for a long time [4, p.124]. The amount of solutions is called the cardinality.

Specific restrictions may be placed on the parts of a composition: a possible restriction may be that all parts must be even. A second-order restriction means that, rather than one specific restriction, there is a list of restrictions that each apply to one specific part: for example, part 1 must be even, part 2 must be between 2 and 20, part 3 must be a prime, etc.. Abramson discusses restricted compositions in his paper on calculating cardinality [1]. The discussed restricted compositions include the subset of compositions which Page calls second-order restricted compositions in his paper in which he introduces an algorithm that generates the full set of solutions [3]. Walsh speaks of $(m_1, ..., m_k)$-bounded weak compositions [6], which are a subset of second-order restricted weak compositions: the restriction on each part i is that it must be in the interval $[0, m_i]$, inclusive. It is this type of composition that this research concerns itself with, as described in section 1.

In relation to the sampling of integer compositions, we use the following definition of uniformity: considering a system where each solution of the solution space has an associated number which describes the probability that this solution is the one which is output by an algorithm, the system is uniform if the associated probability is equal for each solution. Besides demonstrating that this is the case, uniformity can also be proven by performing an equivalence test between the generated samples and a sample set that is already proven to be uniform.

3. MOTIVATION

The value of a solution to the aforementioned problem is that second-order restricted compositions represent a selection of N objects from k distinct sources with limited availability. To illustrate: they can be used to generate random starting populations for natural selection simulations – consider a population of, say, less than 5 wolves, between 10 and 20 sheep, and at least 15 rabbits, with a total population of 50. Having these starting populations generated at random, with each possible starting population being equally likely, allows for unbiased sampling of simulation results without having to execute a simulation for each possible beginning state. One can easily conceive of similar distribution problems where uniform random second-order restricted k-compositions can be applied.

Algorithms have been written which generate all solutions to this constraint problem. However, when the goal is

32nd Twente Student Conference on IT Jan. 31st, 2020, Enschede, The Netherlands.

to generate a single solution, these algorithms are not satisfactory: when generating the full solution space, at least one computation must be made for each solution that exists, and at least one memory location is necessary to store this solution. The amount of solutions i.e. the cardinality of a composition is explored in section 2, but in the worst case, it is $\frac{(N+k-1)!}{N!k!}$.

For small N, k and elements of R, the cardinality can already be very large. It must be possible to generate a single unbiased random solution without requiring the time and memory necessary for generating every solution.

4. APPROACH

There exist algorithms created by Vajnovzski [5], Walsh [6], Page [3], and Opdyke [2], all of which may be used to generate the array of the full solution space of second-order restricted compositions. Because the size of the solution space is subject to combinatorial explosion, generating it in its entirety is undesirable when only a single solution is needed. However, when many samples from the solution space are needed, it may be efficient to calculate and store the full solution space, and sample from it as needed. We implement - or import - at least one of the mentioned algorithms and use it to generate the array of solutions for some N, k and second-order restriction, then take uniform random samples from this array. The uniform distribution of these samples will be the control group, against which the new approach below will be compared.

We now introduce the approach which was taken for this research: a novel algorithm which maps a uniformly random integer to a $(m_1, ..., m_k)$-bounded k-composition of N. This mapping uses the relation between the cardinality of k-compositions with bounds $(m_1, m_2, ..., m_k)$ and the cardinalities of $(k-1)$-compositions with bounds $(m_2, ..., m_k)$: consider the case where we are deciding whether to assign the value 0, 1 or 2 to a_1 in the second-order k-composition of N $(a_1, ..., a_k)$. We calculate that if we decide $a_1 = 0$, then the amount of solutions to the $(k-1)$-composition of $(N - a_1)$ equals 100. We then calculate that if $a_1 = 1$ instead, that amount of solutions becomes 50, and if $a_1 = 2$, then the amount of solutions is 10. Therefore, in 100 cases the random integer maps to a composition with $a_1 = 0$, in 50 cases it maps to a composition with $a_1 = 1$, and in the remaining 10 cases it maps to a composition with $a_1 = 2$. This relation between cardinalities is explored in depth in section 7.1.

Because the mapping is bijective, each valid input from the domain of the function must map to a unique composition. This implies that if we can sample uniformly from the domain, we can use those samples to get uniform samples from the range of the mapping, i.e., the solution set of compositions. Most programming languages natively provide functions for the uniform random generations of integers from an interval, which is sufficient for sampling uniformly from the domain. To verify that the resulting distribution of compositions is uniform, we will compare the results of a χ^2-test of the distribution against the results of a χ^2-test of a distribution which is known to be uniform.

5. PROBLEM STATEMENT

This paper answers the following research question:

Can the relation between cardinalities of second-order restricted compositions be used to create a bijective mapping from an integer to a unique second-order restricted k-composition of N?

Can such a mapping be used to generate uniform random samples for solutions of second-order restricted k-composition of N?

How does the application of such a mapping compare against existing algorithms in terms of execution time and memory usage?

6. EXISTING ALGORITHMS

6.1 Page's Algorithm

Page's algorithm can generate the solution space for compositions with arbitrary second order restrictions - even restrictions such as: a_1 must be in the set $\{1, 5, 9, 12, 15\}$. So it is more powerful than the scope of this research requires. We implemented Page's algorithm and used to generate the $(m_1, ..., m_k)$-restricted compositions of N. For this use case, the algorithm functions as follows:

The algorithm starts with a vector of length k $\langle a_1 = 0, ..., a_k = 0\rangle$ consisting of zeroes: this is a list of all $(m_1, ..., m_k)$-restricted compositions of 0. Next, for each a_i in the vector that is smaller than the corresponding m_i, a copy of the vector is made, and in this copy, a_i is incremented - these copies are collected in a set of vectors: this set is the set of $(m_1, ..., m_k)$-restricted compositions of 1. Next, these collected compositions are incremented to make the set of $(m_1, ..., m_k)$-restricted compositions of 2, then of 3, and so on, up to N.

The issue with this approach is clear: the algorithm loops over the $(m_1, ..., m_k)$-restricted compositions of 0 through N, which is a much larger number than merely the number of $(m_1, ..., m_k)$-restricted compositions of N. As such, the time complexity of this algorithm can be expressed as $O(n \times \binom{n+k}{k})$. Despite this drawback, the algorithm does succeed in generating the full set of $(m_1, ..., m_k)$-restricted compositions of N, thus allowing for random sampling from this solution space.

6.2 Walsh' algorithm

Unlike Page's algorithm, Walsh' algorithm as provided is only applicable for the restricted compositions that this research paper concerns itself with [6]. It proposes a lexicographical ordering for all $(m_1, ..., m_k)$-restricted k-compositions of N where each composition is succeeded by another composition that has some a_i incremented and some a_j decremented by 1. The paper also proposes an algorithm that generates a composition's successor according to this ordering.

The challenge of this method is determining which part should be incremented and which part should be decremented. The solution to this challenge is the usage of some helper arrays to keep track of state, which is used in a number of cascading conditional statements - this means that the algorithm that transforms some ith composition to the $(i+1)$th composition is in constant time. Therefore, the full algorithm that generates all $(m_1, ..., m_k)$-restricted compositions of N has the time complexity $O(\binom{n+k}{k})$. This is a better time complexity than Page's algorithm provides, as well as the best time complexity possible when generating $\binom{n+k}{k}$ elements.

7. NOVEL ALGORITHM

7.1 Relation between cardinalities

Note that in this section, we use different types of second order restrictions for k-compositions $\langle a_1, a_2, ..., a_k\rangle$: these types are upper-bound restrictions of the form $R = \langle m_1, m_2, ..., m_k\rangle$ where each part a_i must be non-negative and less than or equal to m_i, and interval restrictions of

the form $R = \langle [r_1, R_1], [r_2, R_2], ..., [r_k, R, k] \rangle$ with every $r_i \geq 0$ and every $r_i \leq R_i$, where each part a_i must be within the interval $[r_i, R_i]$. An upper-bound restriction m_i is equivalent to an interval restriction of $[0, m_i]$.

Let us define a function $card(n, R)$ with n and R defining a second-order restricted k-composition of n, with $R = \langle m_1, m_2, ..., m_k \rangle$ being the vector of restrictions of length k. The function $card(n, R)$ returns the cardinality of this composition, i.e. the amount of unique vectors $\langle a_1, a_2, ..., a_k \rangle$ that are a valid k-composition of n with second-order restriction R. This cardinality can be found by taking the coefficient of x^n from the following generator function [4]:

$$(1 + x + x^2 + ... + x^{R_1}) \times (1 + x + x^2 + ... + x^{R_2}) \times ... \times (1 + x + x^2 + ... + x^{R_k}) \quad (1)$$

To demonstrate this, consider the composition with $n = 6$ with upper-bound restrictions $R = \langle 3, 4, 5 \rangle$. We can find its cardinality by expanding the following polynomial and taking the coefficient of x^6:

$$(1 + x + x^2 + x^3) \times (1 + x + x^2 + x^3 + x^4) \times (1 + x + x^2 + x^3 + x^4 + x^5) \quad (2)$$

Because we are not interested in the coefficients of x^7 and higher, we can disregard those entirely. We expand the two rightmost polynomials, ignoring coefficients of x^7 and higher:

$$(1 + x + x^2 + x^3) \times (1 + 2x + 3x^2 + 4x^3 + 5x^4 + 5x^5 + 4x^6) \quad (3)$$

To expand further, we calculate the following:

$$\begin{array}{rl} 1 \times (& 1{+}2x{+}3x^2{+}4x^3\ {+}5x^4\ {+}5x^5\ {+}4x^6\ ){+} \\ x \times (& 1\ {+}2x\ {+}3x^2\ {+}4x^3\ {+}5x^4\ {+}5x^5\ ){+} \\ x^2 \times (& 1\ {+}2x\ {+}3x^2\ {+}4x^3\ {+}5x^4\ ){+} \\ x^3 \times (& 1\ {+}2x\ {+}3x^2\ {+}4x^3\ ) \\ = & 1{+}3x{+}6x^2{+}10x^3{+}14x^4{+}17x^5{+}\underline{18x^6} \end{array}$$

So $card(6, \langle 3, 4, 5 \rangle) = 18$. Now note how the polynomial $(1{+}2x{+}3x^2{+}4x^3{+}5x^4{+}5x^5{+}4x^6)$ itself corresponds with the compositions with upper-bound restrictions $\langle 4, 5 \rangle$. We can easily find that $card(6, \langle 4, 5 \rangle) = 4$, $card(5, \langle 4, 5 \rangle) = 5$, $card(4, \langle 4, 5 \rangle) = 5$ and $card(3, \langle 4, 5 \rangle) = 4$. Looking at the table, the following is clear:

$$card(6, \langle 3, 4, 5 \rangle) = card(6, \langle 4, 5 \rangle) + card(5, \langle 4, 5 \rangle) + card(4, \langle 4, 5 \rangle) + card(3, \langle 4, 5 \rangle) \quad (4)$$

This can be generalized to:

$$card(n, \langle m_1, m_2, ..., m_k \rangle) = \Sigma_{i=0}^{m_1} card(n - i, \langle m_2, ..., m_k \rangle) \quad (5)$$

Next, we will show that

$$card(n, \langle m_1, m_2, ..., m_k \rangle) = card(n, \langle 0, m_1, m_2, ..., m_k \rangle) \quad (6)$$

This is because the corresponding polynomial of the upper-bound restriction 0 is $x^0 = 1$. If we want to change the polynomial of the upper-bound restrictions $\langle m_1, m_2, ..., m_k \rangle$ into the polynomial of $\langle 0, m_1, m_2, ..., m_k \rangle$, we must multiply it by 1; thus retaining the original polynomial.

Next, we consider interval restricted compositions; Specifically, interval restricted compositions that correspond with an upper-bound restricted composition — in the sense that for each interval restriction $[r_i, R_i]$, $r_i = 0$ and $R_i = m_i$, except in the restriction on its first part, which is instead an interval containing only one integer: $[r_1, R_1] = [a_1, a_1]$. We will show that the following is true for compositions of this form:

$$card(n, \langle [a_1, a_1], [0, m_2], ..., [0, m_k] \rangle) = card(n + 1, \langle [a_1 + 1, a_1 + 1], [0, m_2], ..., [0, m_k] \rangle) \quad (7)$$

The polynomial of $\langle [a_1, a_1], [0, m_2], ..., [0, m_k] \rangle$ looks like $x^{a_1} \times f(x)$, $f(x)$ being the polynomial of $\langle [0, m_2], ..., [0, m_k] \rangle$. Next, The polynomial of $\langle [a_1 + 1, a_1 + 1], [0, m_2], ..., [0, m_k] \rangle$ looks like $x^{a_1+1} \times f(x)$, or $x \times x^{a_1} \times f(x)$.

Now, if the polynomial $x^{a_1} \times f(x)$ contained the term ax^n, in $x \times x^{a_1} \times f(x)$ this term is multiplied by x, along with all other terms, i.e. $x \times ax^n = ax^{n+1}$. So, if we are looking for the coefficient a, in $x^{a_1} \times f(x)$ we must look for the coefficient of x^n, whereas in $x \times x^{a_1} \times f(x)$ we must look for the coefficient of x^{n+1}.

This has the following consequences:

$$\begin{aligned} card(n, \langle m_1, m_2, ..., m_k \rangle) &= card(n, \langle 0, m_1, m_2, ..., m_k \rangle) \\ &= card(n, \langle [0, 0], [0, m_1], [0, m_2], ..., [0, m_k] \rangle) \\ &= card(n + i, \langle [i, i], [0, m_1], [0, m_2], ..., [0, m_k] \rangle) \end{aligned} \quad (8)$$

Meaning that

$$card(n, \langle m_1, m_2, ..., m_k \rangle) = \Sigma_{i=0}^{m_1} card(n, \langle [i, i], [0, m_2], ..., [0, m_k] \rangle) \quad (9)$$

To give this slightly more semantic meaning: For a k-composition of n $\langle a_1, a_2, ..., a_k \rangle$ with upper-bound restrictions $\langle m_1, m_2, ..., m_k \rangle$, the following is true:

$$card(n, \langle m_1, m_2, ..., m_k \rangle) = \Sigma_{i=0}^{m_1} card(n, \langle m_1, m_2, ..., m_k \rangle | a_1 = i) \quad (10)$$

Looking back at our example of the composition $\langle a_1, a_2, a_3 \rangle$ of 6 with upper-bound restrictions $\langle 4, 5, 6 \rangle$, the following statements are true:

$$card(6, \langle 4, 5 \rangle) = card(6, \langle 3, 4, 5 \rangle | a_1 = 0) = 4$$

$$card(5, \langle 4, 5 \rangle) = card(6, \langle 3, 4, 5 \rangle | a_1 = 1) = 5$$

$$card(4, \langle 4, 5 \rangle) = card(6, \langle 3, 4, 5 \rangle | a_1 = 2) = 5$$

$$card(3, \langle 4, 5 \rangle) = card(6, \langle 3, 4, 5 \rangle | a_1 = 3) = 4$$

And of course

$$\begin{aligned} card(6, \langle 3, 4, 5 \rangle) &= card(6, \langle 3, 4, 5 \rangle | a_1 \in [0, 3]) \\ &= card(6, \langle 3, 4, 5 \rangle | a_1 = 0) \\ &+ card(6, \langle 3, 4, 5 \rangle | a_1 = 1) \\ &+ card(6, \langle 3, 4, 5 \rangle | a_1 = 2) \\ &+ card(6, \langle 3, 4, 5 \rangle | a_1 = 3) \end{aligned} \quad (11)$$

We can verify this by looking at the full list of compositions:

$a_1 = 0$	$a_1 = 1$	$a_1 = 2$	$a_1 = 3$
$\langle 0,1,5\rangle$	$\langle 1,0,5\rangle$	$\langle 2,0,4\rangle$	$\langle 3,0,3\rangle$
$\langle 0,2,4\rangle$	$\langle 1,1,4\rangle$	$\langle 2,1,3\rangle$	$\langle 3,1,2\rangle$
$\langle 0,3,3\rangle$	$\langle 1,2,3\rangle$	$\langle 2,2,2\rangle$	$\langle 3,2,1\rangle$
$\langle 0,4,2\rangle$	$\langle 1,3,2\rangle$	$\langle 2,3,1\rangle$	$\langle 3,3,0\rangle$
	$\langle 1,4,1\rangle$	$\langle 2,4,0\rangle$	

7.2 Mapping an integer to a composition

We can use the properties described in the previous subsection to create a function that maps each integer in the range $[1, card(n, R)]$ to a unique composition of n with upper-bound restrictions R. This is desirable because it means that, if we can sample integers from this interval uniformly, we can also generate compositions from the solution space that are uniformly distributed. Using Walsh' method, this is already possible, but it requires the generation of the full solution space. For this method, this is not necessary.

Consider the same composition as before: the 3-composition of 6, $\langle a_1, a_2, a_3\rangle$, with upper-bound restrictions $\langle 3,4,5\rangle$. This composition has a cardinality of 18. We randomly pick a number i between 1 and 18, inclusive: say this number $i = 11$. We must first determine which value for a_1 to pick.

	Cardinality	Cumulative Sum
$card(6, \langle 3,4,5\rangle \mid a_1 = 0)$	4	4
$card(6, \langle 3,4,5\rangle \mid a_1 = 1)$	5	9
$card(6, \langle 3,4,5\rangle \mid a_1 = 2)$	5	14
$card(6, \langle 3,4,5\rangle \mid a_1 = 3)$	4	18

If i is within the bounds of the first cumulative sum, i.e., $i \leq 4$, we determine that $a_1 = 0$. If it is between the first and second cumulative sums, i.e., $4 < i \leq 9$, then $a_1 = 1$. Following this pattern, we conclude that $a_1 = 2$. We now have a situation where we must find a composition of 6 with the following interval constraints: $\langle [2,2], [0,4], [0,5]\rangle$. This is equivalent to finding a composition of 4 with the upper-bound constraints $\langle 4,5\rangle$, which we can do using the same method as before. For our new i, we take the old value of i and subtract the highest cumulative sum lower than that i, i.e. $i_{new} = 11 - 9 = 2$.

	Cardinality	Cumulative Sum
$card(4, \langle 4,5\rangle \mid a_2 = 0)$	1	1
$card(4, \langle 4,5\rangle \mid a_2 = 1)$	1	2
$card(4, \langle 4,5\rangle \mid a_2 = 2)$	1	3
$card(4, \langle 4,5\rangle \mid a_2 = 3)$	1	4
$card(4, \langle 4,5\rangle \mid a_2 = 4)$	1	5

i_{new} is equal to the second cumulative sum, so we pick $a_2 = 1$. It follows that $a_3 = 3$, because $a_2 + a_3 = 4$. This gives us the composition $\langle 2,1,3\rangle$ for $i = 11$.

7.3 Time and Space complexity

This new method requires two steps: creating the cardinality matrix generating the desired solution.

The cardinality matrix is a matrix of size $k \times n$: There is one row for each part of the composition, and each row i contains the coefficients of the expanded polynomials of $a_i \times ... \times a_k$. Each cell of this matrix can be filled in constant time (assuming the addition and subtraction of integers is possible in constant time in the language). This means that the cardinality matrix can be generated in $O(n \times k)$, and the required space is $k \times n \times$ the size of an integer. For large inputs, it may be necessary to use integers or arbitrary size, such as Java's BigInteger, to avoid overflow.

The second part of the algorithm, which generates the solution, has a loop over N. Within this loop is another loop, which is executed at most k times over the course of the algorithm, giving this algorithm a time complexity of $(N + k)$. The memory requirements, in addition to the memory of the cardinality matrix which is an input, is the memory needed to store a single composition.

Algorithm 1: Cardinality Matrix Initialization

input : integer N, integer vector $[m_1, ..., m_k]$
output : M = cardinality matrix of $(m_1, ..., m_k)$-bounded k-compositions of N

initialize M with k rows and N columns;
set elements 0 to m_k of the bottom row of M to 1;
set the remaining elements of the bottom row of M to 0;
for $i \leftarrow k-1$ **to** 0 **do**
  $M_{i,0} \leftarrow 1$;
  for $j \leftarrow 1$ **to** N **do**
    $summand \leftarrow M_{i+1,j}$;
    if $j > m_i$ **then**
      $subtrahend \leftarrow M_{i+1,j-1-m_i}$;
    else
      subtrahend $\leftarrow 0$;
    end
    $M_{i,j} \leftarrow M_{i,j-1} - subtrahend + summand$;
  end
end
return *matrix* ;

Algorithm 2: Mapping Integer to Composition

Data: cardinality matrix M, solution number x
Result: The unique $(m_1, ..., m_k)$-bounded k-composition of N corresponding with solution number

$k \leftarrow M.rows$;
$n \leftarrow M.columns$;
$row \leftarrow 1$;
$col \leftarrow M.columns$;
for $col \leftarrow M.columns$ **to** 0 **do**
  while $row < k$ *and* $x \leq M_{row,col}$ *and* $M_{row,col} \neq 0$ **do**
    $a_{row} \leftarrow n - col$;
    $n \leftarrow n - a_{row}$;
    $row \leftarrow row + 1$;
  end
  $x \leftarrow M_{row,col}$;
end
$a_k \leftarrow n$;

8. VERIFICATION

8.1 Correctness

We tested the correctness of the algorithm empirically by testing, for $N \in [1, 25]$ and $k \in [1, 10]$ and with random selections for $\langle m_1, ..., m_k\rangle$, whether each number in the range $[1, card(N, \langle m_1, ..., m_k\rangle)]$ mapped to a unique $(m_1, ..., m_k)$-bounded k-composition of N.Because the solution space was too large to compute within reasonable time and memory constraints, we did not perform this test for higher values of N and k.

For higher values of N and k, we tested the following: we sampled 10000 random solutions and tested whether the solutions were indeed correct, i.e., whether they added up to N. We also specifically calculated solution 1 and solution $card(N, \langle m_1, ..., m_k\rangle)$, and in addition to testing

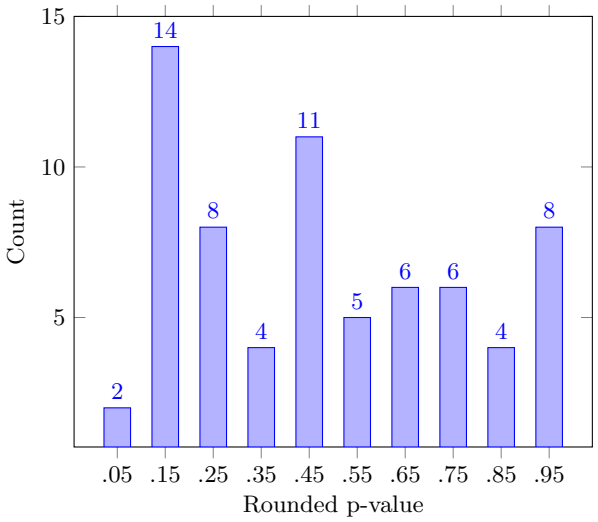

Figure 1. Counts of p-values for similarity tests between sample set generated with the new algorithm and sample set uniformly sampled from the solution space

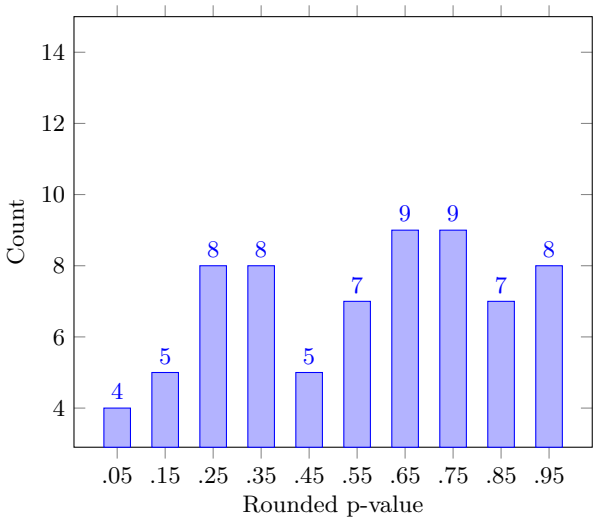

Figure 2. Counts of p-values for similarity tests between sample sets both uniformly sampled from the solution space

their correctness, we also tested whether those are indeed the first and last solutions.

These tests all passed for approximately $N \leq 100, k \leq 20$, at which point the generation of the cardinality matrix failed due to the overflow of 64-bit integers. We are confident that the use of integers of arbitrary size over 64-bit integers is sufficient to let tests pass for Ns and ks of arbitrary size.

8.2 Uniformity

To test the uniformity of the newly introduced algorithm, we performed a number of χ^2-tests. Each test was performed using the following method: a random N between 1 and 20 is selected, a random k between 2 and 10 is selected, and the restriction vector $\langle m_1, ..., m_k \rangle$ is generated with each m_i randomly selected from 1 to $min(N, 11)$. These limits are chosen to allow for a wide range of compositions, while having a reasonably low chance to have a cardinality higher than the maximum size of an array in Java - any composition with a cardinality that surpassed this number is discarded.

For each of these compositions, the full solution space is generated, and we take $1000 \times card(N, \langle m_1, ..., m_k \rangle)$ uniform samples from the solution space, which is generated using Page's algorithm. We also take $1000 \times card(N, \langle m_1, ..., m_k \rangle)$ uniform samples from the range $[1, card(N, \langle m_1, ..., m_k \rangle)]$, which are then map to compositions using the new algorithm. A χ^2-test is then used to test whether the two sample sets come from the same distribution. The resulting p-values from these tests are shown in figure 1, rounded to two decimals.

While some p-values are outside the confidence interval of 90%, this is to be expected from such a test, and does not imply that the two distributions are dissimilar. To show this, we also perform χ^2-tests to test whether two sample sets uniformly sampled from the solution space come from the same distribution, which is known to be true. These results are found in figure 2. This same test is done on two sample sets generated by sampling uniformly from the range $[1, card(N, \langle m_1, ..., m_k \rangle)]$ and mapping those numbers to compositions using the new algorithm. These results are found in figure 3. The results from figures 1, 2

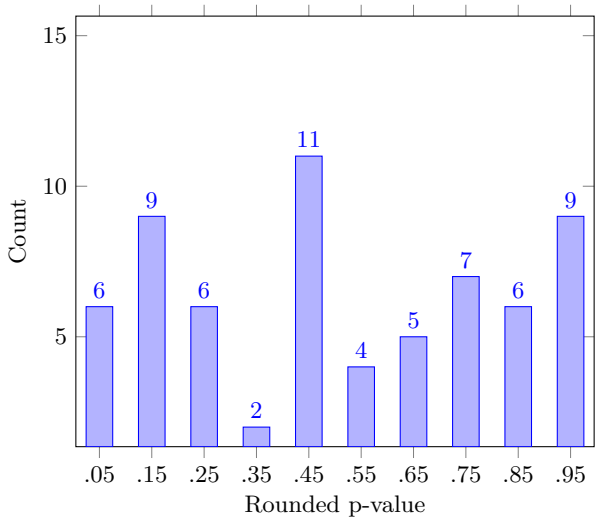

Figure 3. Counts of p-values for similarity tests between sample sets both generated with the new algorithm

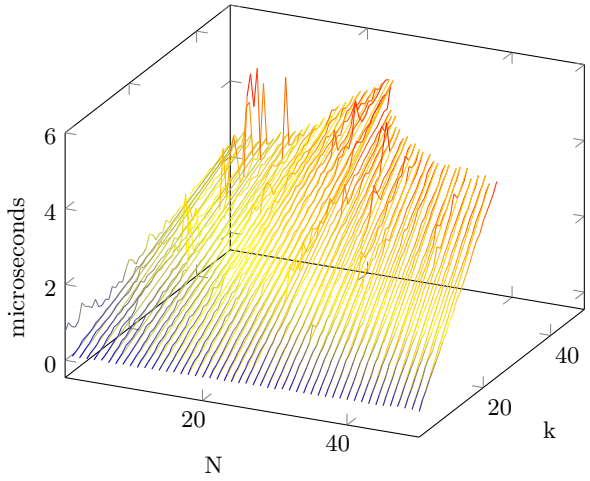

Figure 4. Execution times the generation of a random composition using the new algorithm

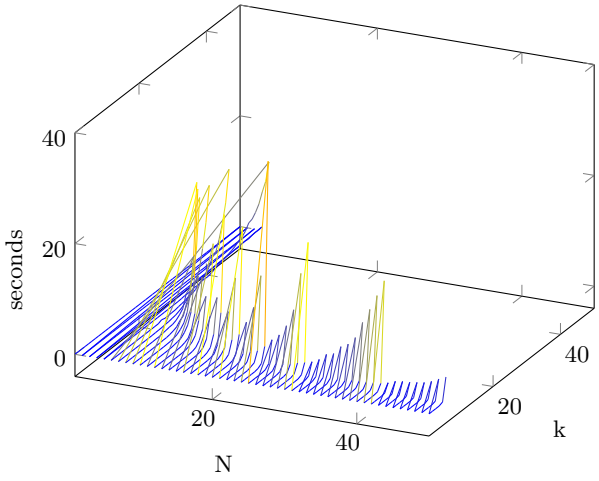

Figure 5. Execution times the generation of a random composition using the Page's algorithm

and 3 are similar enough to be confident that they imply the same thing: that the sample sets in the χ^2-tests come from similar distributions, i.e. uniform distributions.

8.3 Time Complexity

We compared the time that the algorithm spends to generate a single random composition for N and k up to 50. In these benchmarks, each part of $\langle m_1, ..., m_k \rangle$ is equal to N: this causes $card(N, \langle m_1, ..., m_k \rangle)$ to be at the maximum for that N and k. Results for this can be found in figure 4. For each pair (N, k), parts 1 and 2 of the algorithm were executed 100000 times, and the average time was taken for each pair.

There is a cutoff point around 6 microseconds: beyond this point the cardinality matrix could not be created because of 64-bit integer overflows. Despite this, the figure shows that time increases linearly with N and with k.

Figure 5 shows the time spent to generate a single random composition using Page's algorithm. The time grows rapidly with the increase of N and k - there is not enough data available to analyze the nature of this growth, because the algorithm runs out of memory space, but the difference in performance is clear when the labels of the z-axes of figures 4 and 5 are compared: figure 4 is measured in microseconds, while figure 5 is measured in seconds.

9. CONCLUSION

There now exists an algorithm that can be used to map an integer to a unique $(m_1, , , , , m_k)$-restricted composition of N, as described in the paper. The mapping is a bijection - each composition is pointed to by a unique integer between 1 and the number of compositions. By mapping uniform random numbers from the interval $[1, card(N, \langle m_1, ..., m_k \rangle)]$ to compositions using this algorithm, we can generate uniform random $(m_1, , , , , m_k)$-restricted compositions of N. This algorithm is significantly more efficient with regards to time and space complexity than the previously available methods: it is much faster to generate only one composition, instead of each possible one.

10. REFERENCES

[1] M. Abramson. Restricted combinations and compositions. *Fibonacci Quart*, 14(5):439–452, 1976.

[2] J. D. J. Opdyke. A unified approach to algorithms generating unrestricted and restricted integer compositions and integer partitions. *Journal of Mathematical Modelling and Algorithms*, 9(1):53–97, Mar 2010.

[3] D. R. Page. Generalized algorithm for restricted weak composition generation. *Journal of Mathematical Modelling and Algorithms in Operations Research*, 12(4):345–372, Dec 2013.

[4] J. Riordan. *Introduction to combinatorial analysis*. Courier Corporation, 1958.

[5] V. Vajnovszki. Generating permutations with a given major index. *arXiv preprint arXiv:1302.6558*, 2013.

[6] T. Walsh. Loop-free sequencing of bounded integer compositions. *JCMCC. The Journal of Combinatorial Mathematics and Combinatorial Computing*, 33, 01 2000.