Numerical Algorithm for Pólya Enumeration Theorem

CONRAD W. ROSENBROCK, WILEY S. MORGAN, and GUS L. W. HART,
Brigham Young University
STEFANO CURTAROLO, Duke University
RODNEY W. FORCADE, Brigham Young University

Although the Pólya enumeration theorem has been used extensively for decades, an optimized, purely numerical algorithm for calculating its coefficients is not readily available. We present such an algorithm for finding the number of unique colorings of a finite set under the action of a finite group.

Categories and Subject Descriptors: G.2.1 [**Discrete Mathematics**]: Combinatorics

General Terms: Combinatorial Algorithms, Counting Problems

Additional Key Words and Phrases: Pólya enumeration theorem, expansion coefficient, product of polynomials

ACM Reference Format:
Conrad W. Rosenbrock, Wiley S. Morgan, Gus L. W. Hart, Stefano Curtarolo, and Rodney W. Forcade. 2016. Numerical algorithm for pólya enumeration theorem. J. Exp. Algorithmics 21, 1, Article 1.11 (August 2016), 17 pages.
DOI: http://dx.doi.org/10.1145/2955094

1. INTRODUCTION

A circle partitioned into 4 equal sectors can be colored 16 different ways using two colors, $2^4 = 16$, as shown in Figure 1. But only 6 of these colorings are symmetrically distinct, several others being equivalent (under rotations and reflections) as shown by the arrows in the figure. The Pólya enumeration theorem provides a way to determine how many symmetrically distinct colorings there are with, for example, all sectors red (only one, as shown in the figure), one red sector and three green (again, only one), or the number with two red sectors and two green sectors (two, as shown in the figure). Borrowing a word from physics and chemistry, we refer to the partition of red and green sectors as the *stoichiometry*. For example, a coloring with 1 red sector and 3 green sectors has a stoichiometry of 1:3.

The Pólya theorem [Pólya 1937; Pólya and Read 1987] produces a polynomial (generating function), shown in the figure, whose coefficients answer the question of how many distinct colorings there are for each stoichiometry (each partition of the colors). For example, the $2r^2g^2$ term in the polynomial indicates that there are two distinct ways to color the circle with 2:2 stoichiometry (). For all other stoichiometries (4:0,

This work was supported under Grant No. ONR (MURI N00014-13-1-0635).
Authors' addresses: C. W. Rosenbrock, W. S. Morgan, and G. L. W. Hart, Department of Physics and Astronomy, 84602, Brigham Young University; S. Curtarolo, Materials Science, Electrical Engineering, Physics and Chemistry, 27708, Duke University; R. W. Forcade, Department of Mathematics, 84602, Brigham Young University.

DOI: http://dx.doi.org/10.1145/2955094

$$P(r,g) = r^4 + r^3g + 2r^2g^2 + rg^3 + g^4$$

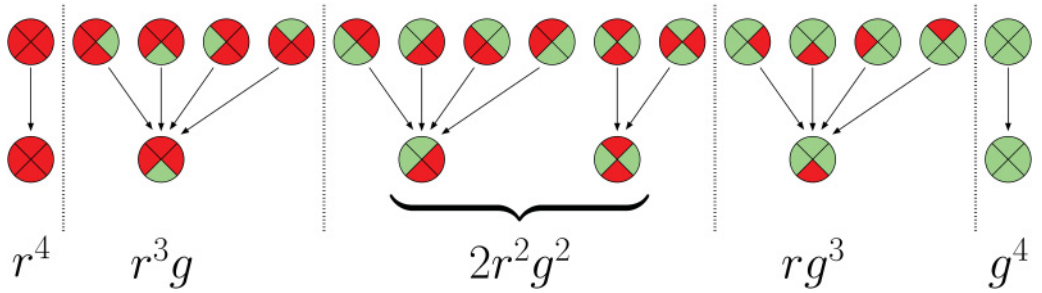

Fig. 1. Top row: All possible two-color colorings of a circle divided into four equal sectors (left side of figure). Bottom row: All symmetrically distinct binary colorings of the circle. Arrows indicate combinatorically distinct colorings that are equivalent by symmetry.

0:4, 1:3, and 3:1), the polynomial coefficients are all 1, indicating that for each of these cases there is only one distinct coloring, as is obvious from the figure.

A common problem in many fields involves enumerating[1] the *symmetrically distinct* colorings of a finite set, similar to the toy problem of Figure 1. The Pólya theorem has shown its wide range of applications in a variety of contexts. Classically, it was applied to counting chemical isomers [Robinson et al. 1976; Kennedy et al. 1964; Pólya 1937] and graphs [Harary 1955]. Recent examples include confirming enumerations of molecules in bioinformatics and chemoinformatics [Deng and Qian 2014; Ghorbani and Songhori 2014]; unlabeled, uniform hypergraphs in discrete mathematics [Qian 2014]; analysis of tone rows in musical composition [Lackner et al. 2015]; commutative binary models of Boolean functions in computer science [Genitrini et al. 2015]; generating functions for single-trace-operators in high-energy physics [McGrane et al. 2015]; investigating the role of nonlocality in quantum many-body systems [Tura et al. 2015]; and photosensitizers in photosynthesis research [Taniguchi et al. 2014].

In computational materials science, chemistry, and related subfields such as computational drug discovery, combinatorial searches are becoming increasingly important, especially in high-throughput studies [Curtarolo et al. 2013]. As computational methods gain a larger market share in materials and drug discovery, algorithms such as the one presented in this article are important as they provide validation support to complex enumeration codes. Pólya's theorem is the only way to independently confirm that an enumeration algorithm has performed correctly. The present algorithm has been useful in checking a new algorithm extending the work in Hart and Forcade [2008, 2009] and Hart et al. [2012], and Pólya's theorem was recently used in a similar crystal enumeration algorithm [Mustapha et al. 2013] that has been incorporated into the CRYSTAL14 software package [Dovesi et al. 2014].

Despite the widespread use of Pólya's theorem in different science and mathematics contexts, a low-level, numerical implementation is not available. Typical approaches use Computer Algebra Systems (CASs) to symbolically generate the Pólya polynomial. This strategy is ineffective for two reasons. First, CASs are too slow for large problems that arise in a research setting, and, second, generating the entire Pólya polynomial (which can have billions or trillions of terms) is unnecessary when one is interested in only a single stoichiometry.

[1]The Pólya theorem does not generate the list of unique colorings (which is generally a much harder problem), but it does determine the *number* of unique colorings.

Here we demonstrate a low-level algorithm for finding the polynomial coefficient corresponding to a single stoichiometry. It exploits the properties of polynomials and *a priori* knowledge of the relevant term. We briefly describe the Pólya enumeration theorem in Section 2, followed by the algorithm for calculating the polynomial coefficients in Section 3. In the final section, we investigate the scaling and performance of the algorithm.

2. PÓLYA ENUMERATION THEOREM

2.1. Introduction the Pólya Enumeration Theorem

Pólya's theorem provides a simple way to construct a generating polynomial whose coefficients count the numbers of symmetrically distinct colorings for each possible stoichiometry. The polynomial in Figure 1 above was easy to verify because we were able to hand count the symmetrically distinct colorings. But suppose there were dozens of colors and dozens of sites to be colored and hundreds of symmetries to apply. In that case, it is easier to use Pólya's theorem to construct the polynomial directly from the symmetry group.

To describe this very useful theorem, we refer once more to Figure 1. There are four symmetries—the identity, two 90° rotations (clockwise and counterclockwise), and a 180° rotation. If we label the colorable sectors 1, 2, 3, and 4, and write the permutations in *disjoint-cycle* notation, we have (1)(2)(3)(4) for the identity, the two 90° rotations are represented by (1234) and (1432), while the 180° rotation is (13)(24) in cycle notation.

Now Pólya's theorem simply tells us to replace each cycle of length λ with a sum of λ-th powers of variables corresponding to the colors available. For example, letting r and g stand for red and green, the identity is represented by $(r+g)(r+g)(r+g)(r+g)$, the two 90° rotations are each replaced by (r^4+g^4), and the 180° rotation is replaced by $(r^2+g^2)(r^2+g^2)$. When we *average* these four polynomials, we get the Pólya polynomial predicted above:

$$\begin{aligned} P(r,g) &= \frac{1}{4}\big((r+g)(r+g)(r+g)(r+g) + (r^4+g^4) + (r^4+g^4) + (r^2+g^2)(r^2+g^2)\big) \\ &= r^4 + r^3g + 2r^2g^2 + rg^3 + g^4. \end{aligned} \tag{1}$$

In other words, Pólya's theorem relies on a structural representation of the symmetries *as permutations written in disjoint-cycle notation* to construct the generating polynomial we need.

The problem with Pólya, however, is that it requires us to compute the *entire* polynomial when we may need only one of its coefficients. For example, if we have 50 sites to color, and 20 colors available, the number of *terms* in our polynomial (regardless of symmetries) would be about 4.6×10^{16}. That is a lot of work (and memory) to compute the entire polynomial (and all of those very large terms) if we needed *only* to know the number of symmetrically distinct colorings for a single stoichiometry. That information is contained in just 1 term of the 46 quadrillion terms of the Pólya polynomial. Can we spare ourselves the work of computing all the others?

Suppose we have a target stoichiometry $[c_1 : c_2 : \cdots : c_\xi]$, where ξ is the number of colors and $\sum_{j=1}^{\xi} c_j = n$ is the number of sites to be colored. To find the number of symmetrically distinct colorings with those frequencies, we must determine the coefficient of the single term in the Pólya polynomial containing the product $x_1^{c_1}x_2^{c_2}\ldots x_\xi^{c_\xi}$. The Pólya polynomial is the average,

$$P(x_1, x_2, \ldots, x_\xi) = \frac{1}{|G|}\left(\sum_{\pi\in G} P_\pi(x_1, x_2, \ldots, x_\xi)\right), \tag{2}$$

of the polynomials $P_\pi(x_1, x_2, \ldots, x_\xi)$ computed for each permutation π in the symmetry group G, each P_π being formed by multiplying the representations of each disjoint cycle in π (as illustrated in Equation (1)).

Clearly, if we are only interested in the coefficient of $x_1^{c_1}x_2^{c_2}\ldots x_\xi^{c_\xi}$ in P, we may simply find the coefficient of that product in each P_π and add those partial coefficients together. Thus, given a permutation π with k_1 cycles of length r_1, k_2 cycles of length r_2, and so on, up to k_t cycles of length r_t, with $\sum_{i=1}^{t} r_i k_i = n$ (the number of sites, t is the number of cycle types), we must compute the coefficient of $x_1^{c_1}x_2^{c_2}\ldots x_\xi^{c_\xi}$ in P_π.

It is well known that a product of sums is equal to the sum of all products one can obtain by taking one summand from each factor (generalizing the familiar First Outer Inner Last (FOIL) rule used by undergrads to multiply two binomials). Thus the polynomial P_π is the sum of all products of the form $\prod_s x_{i_s}^{\lambda(s)}$ (where the product runs over all cycles s, $\lambda(s)$ is the length of the cycle s, and x_{i_s} is one of the colors chosen from the sum for that cycle). Thus the product we want, $x_1^{c_1}x_2^{c_2}\ldots x_\xi^{c_\xi}$, has a coefficient that simply counts the number of products of the form $\prod_s x_{i_s}^{\lambda(s)}$ where the sum of the exponents for each x_i is equal to the target c_i.

Each cycle, of length r_i ($i = 1 \ldots t$), gets assigned to one of the colors. Let s_{ij} be the number of cycles of length r_i assigned to color j ($j = 1 \ldots \xi$). This defines a $t \times \xi$ matrix $S = (s_{ij})$ of non-negative integers, where (1) the sum of row i equals the number of cycles of length r_i:

$$\sum_{j=1}^{\xi} s_{ij} = k_i \quad \text{(row sum condition)}, \tag{3}$$

and (2) weighted sum of column j must equal the target frequency of the j-th color:

$$\sum_{i=1}^{t} r_i s_{ij} = c_j \quad \text{(column sum condition)}, \tag{4}$$

in order to achieve our target stoichiometry.

For each such matrix, there are a number of possible ways to assign colors to the cycles, with multiplicities prescribed by S. The number is

$$F(S) = \prod_{i=1}^{t} \binom{k_i}{s_{i1}, s_{i2}, \ldots, s_{i\xi}}, \tag{5}$$

the product of the number of ways to do it for each cycle. Thus we are obliged to sum the function $F(S)$, so computed, over all matrices S meeting the given row and column sum conditions (3) and (4).

If we do this computation for each permutation π, and average them (add them and divide by $|G|$), we then get the coefficient of the Pólya polynomial $P(x_1, x_2, \ldots, x_i)$ corresponding to our target stoichiometry $[c_1 : c_2 : \cdots : c_\xi]$. This calculation depends only on the *cycle type* of the permutation, the number of disjoint cycles of different lengths comprising the disjoint-cycle representation. Thus we only need to make an inventory of the cycle types for our permutations and do the calculation once for each distinct cycle type. There will not be more such cycle types than the number of conjugacy classes in the symmetry group. Also, note, the utility of multinomial coefficients in this context stems from the likelihood that our permutations will have many cycles of the same length.

Algorithmically, the process is straight forward. First, we must find all matrices S which meet the row and sum conditions (3) and (4) above. For each successful matrix,

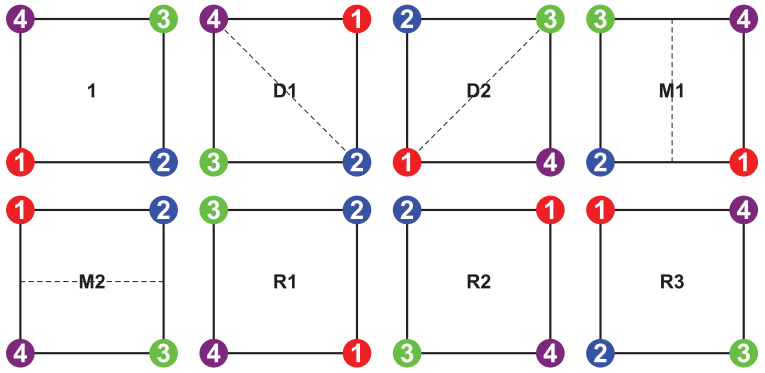

Fig. 2. The symmetry group operations of the square. This group is known as the dihedral group of degree 4 or D_4. The dashed lines are guides to the eye for the horizontal, vertical, and diagonal reflections (**M1**,**M2** and **D1**, **D2**).

we then compute the product of row-multinomial-coefficients. We add those up and multiply by the number of permutations in the conjugacy class, sum those results for the conjugacy classes, and divide by the group order. That gives us the Pólya coefficient for the given stoichiometry.

For example, suppose our permutation is made up of two 1-cycles, three 2-cycles, and one 4-cycle (so the number of sites is 12), and we have three colors with frequencies (red:green:blue $\rightarrow$ 4:6:2) respectively. Then we are looking for 3×3 matrices S whose rows sum to $\begin{pmatrix} 2 \\ 3 \\ 1 \end{pmatrix}$ and whose columns (when dotted with the cycle lengths $\begin{pmatrix} 1 \\ 2 \\ 4 \end{pmatrix}$) sum to 4, 6, and 2 respectively. There are exactly five such matrices (see Figure 3 and discussion in Section 3):

$$\begin{pmatrix} 0 & 0 & 2 \\ 0 & 3 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 2 \\ 2 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 2 & 0 \\ 0 & 2 & 1 \\ 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 2 & 0 \\ 2 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 2 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix}. \tag{6}$$

The multinomial coefficient for the top and bottom row in each case is $\binom{2}{2,0,0} = \binom{2}{2} = 1 = \binom{1}{1,0,0}$, so the $F(S)$ in each case is equal to the multinomial coefficient of the middle row; thus $\binom{3}{3} = 1$ in the first case, $\binom{3}{2,1} = 3$ for the middle three matrices, and $\binom{3}{1,1,1} = 6$ for the right-hand matrix. So our count for this problem is $1 + 3 + 3 + 3 + 6 = 16$. We may check this by computing $(r + g + b)^2(r^2 + g^2 + b^2)^3(r^4 + g^4 + b^4)$ (a la Pólya) and noting that the coefficient of $r^4g^6b^2$ is indeed 16.

Clearly, we can do that for each permutation in the group and sum the results. That is equivalent to determining in how many ways we may assign a single color to each cycle in the permutation—in such a way that the total number of occurrences of each color achieves its target frequency.

2.2. Example: Applying Pólyas Theorem

Here we present a simple example showing how Pólya's theorem is applied to a small, finite group. The square has the set of symmetries displayed in Figure 2. These symmetries include four rotations (by 0°, 90°, 180°, and 270°; labeled **1**, **R1**, **R2**, and **R3**) and four reflections (one horizontal, one vertical, and two for the diagonals; labeled

Table I. Disjoint-Cyclic Form for Each Group Operation in D_4 and the Corresponding Polynomials, Expanded Polynomials and the Coefficient of the x^2y^2 Term for Each

Op.	Disjoint-Cyclic	Polynomial	Expanded	Coeff.
$\mathbb{1}$	(1)(2)(3)(4)	$(x+y)^4$	$x^4+4x^3y+\boxed{6x^2y^2}+4xy^3+y^4$	6
D1	(1, 3)(2)(4)	$(x^2+y^2)(x+y)^2$	$x^4+2x^3y+\boxed{2x^2y^2}+2xy^3+y^4$	2
D2	(1)(2, 4)(3)	$(x^2+y^2)(x+y)^2$	$x^4+2x^3y+\boxed{2x^2y^2}+2xy^3+y^4$	2
M1	(1, 2)(3, 4)	$(x^2+y^2)^2$	$x^4+\boxed{2x^2y^2}+y^4$	2
M2	(1, 4)(2, 3)	$(x^2+y^2)^2$	$x^4+\boxed{2x^2y^2}+y^4$	2
R1	(1, 4, 3, 2)	(x^4+y^4)	$x^4+\quad\quad+y^4$	0
R2	(1, 3)(2, 4)	$(x^2+y^2)^2$	$x^4+\boxed{2x^2y^2}+y^4$	2
R3	(1, 2, 3, 4)	(x^4+y^4)	$x^4+\quad\quad+y^4$	0

M1, **M2** and **D1**, **D2**). This group is commonly known as the dihedral group of degree four, or D_4 for short.[2]

The group operations of the D_4 group can be written in disjoint-cyclic form as in Table I. For each r-cycle in the group, we can write a polynomial in variables x_i^r for $i = 1 \ldots \xi$, where ξ is the number of colors used. For this example, we will consider the situation where we want to color the four corners of the square with only two colors. In that case we end up with just two variables x_1, x_2, which are represented as x, y in the table.

The Pólya representation for a single group operation in disjoint-cyclic form results in a product of polynomials that we can expand. For example, the group operation **D1** has disjoint-cyclic form $(1, 3)(2)(4)$ that can be represented by the polynomial $(x^2 + y^2)(x + y)(x + y)$, where the exponent on each variable corresponds to the length of the r-cycle of which it is a part. For a general r-cycle, the polynomial takes the form

$$\left(x_1^r + x_2^r + \cdots + x_\xi^r\right), \tag{7}$$

for an enumeration with ξ colors. As described in Section 2.1, we exchange the group operations acting on the set for polynomial representations that obey the familiar rules for polynomials.

We will now pursue our example of the possible colorings on the four corners of the square involving two of each color. Excluding the symmetry operations, we could come up with $\binom{4}{2} = 6$ possibilities, but some of these are equivalent by symmetry. The Pólya theorem counts how many *unique* colorings we should recover. To find that number, we look at the coefficient of the term corresponding to the overall color selection (in this example, two of each color); thus we look for coefficients of the x^2y^2 term for each group operation. These coefficient values are listed in Table I. The sum of these coefficients, divided by the number of operations in the group, gives the total number of unique colorings under the entire group action, in this case $(6 + 2 + 2 + 2 + 2 + 0 + 2 + 0)/8 = 16/8 = 2$.

Next, we apply the procedure discussed in connection with Equation (6) to construct the matrix S for one of the permutations of the square. It illustrates the idea behind the general algorithm presented in the next section.

In the symmetries of the square, there is a cycle type consisting of two 1-cycles and one 2-cycle. The two permutations with that type are (1)(3)(24) and (2)(4)(13). The cycle lengths are 1 (with multiplicity 2) and 2 (with multiplicity 1). So each of those

[2]The dihedral groups have multiple, equivalent names. D_4 is also called Dih_4 or the dihedral group of *order* 8 (D_8).

permutations requires a matrix $S = \left(\begin{smallmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{smallmatrix}\right)$ satisfying $s_{11} + s_{12} = 2$ and $s_{21} + s_{22} = 1$ (row sum condition (3)) and $s_{11} + 2s_{21} = 2$ and $s_{12} + 2s_{22} = 2$ (column sum condition (4)). There are only two matrices of non-negative integers satisfying those conditions simultaneously:

$$\begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \text{ and } \begin{pmatrix} 0 & 2 \\ 1 & 0 \end{pmatrix}. \tag{8}$$

For each of these matrices, the row-multinomial coefficients are $\binom{2}{0,2} = 1$ and $\binom{1}{0,1} = 1$ so each matrix yields a product 1. Thus each permutation of this cycle type contributes 2 to the sum. This corresponds to the fact that the coefficient of x^2y^2 in $(x+y)^2(x^2+y^2)$ is 2.

Since there are two permutations of this cycle type, the total contribution of the cycle type to the overall Pólya polynomial is 4 (which must then be divided by the number of symmetries in the group).

Thus, in general, the only problem is to find an efficient way of generating these matrix solutions. Since the problem is equivalent to enumerating all lattice points within a high-dimensional polytope, we presume that a tree search (implemented recursively or via a backtracking algorithm) may be the most efficient way to achieve this.

3. COEFFICIENT-FINDING ALGORITHM

Our implementation of the tree search is fundamentally identical to the method of the last section; however, the details may not be immediately recognizable as such.[3] In this section we rephrase the row and column sum conditions (3) and (4) to highlight the logical connections between our specific implementation and the general ideas from Section 2. We adopt this approach because (1) for pedagogical value, the matrix approach is much easier to visualize and (2) the algorithms presented here mirror the accompanying code closely, which we consider valuable.

First, for a generic polynomial

$$\left(x_1^r + x_2^r + \cdots + x_\xi^r\right)^d, \tag{9}$$

the exponents of each x_i in the *expanded* polynomial are constrained to the set

$$V = \{0, r, 2r, 3r, \ldots, dr\}. \tag{10}$$

Next, we consider the terms in the expansion of the polynomial:

$$\left(x_1^r + x_2^r + \cdots + x_\xi^r\right)^d = \sum_{k_1,k_2,\ldots,k_\xi} \mu_k \prod_{i=1}^{\xi} x_i^{rk_i}, \tag{11}$$

where the sum is over all possibles sequences $k_1, k_2, \ldots, k_\xi$ such that the sum of the exponents (represented by the sequence in k_i) is equal to d,

$$k_1 + k_2 + \cdots + k_\xi = d. \tag{12}$$

[3]If all you are looking for is a working code, you now know enough to use it. Download it at https://github.com/rosenbrockc/polya.

As described in the introduction, the coefficients μ_k in the polynomial expansion Equation (11) are found using the multinomial coefficients

$$\begin{aligned}\mu_k &= \binom{n}{k_1, k_2, \ldots, k_\xi} = \frac{n!}{k_1! k_2! \cdots k_\xi!} \\ &= \binom{k_1}{k_1}\binom{k_1+k_2}{k_2}\cdots\binom{k_1+k_2+\cdots+k_\xi}{k_\xi} \\ &= \prod_{i=1}^{\xi}\binom{\sum_{j=1}^{i} k_j}{k_i}. \end{aligned} \tag{13}$$

Finally, we define the polynomial (7) for an arbitrary group operation $\pi \in \mathbf{G}$ as[4]

$$P_\pi(x_1, x_2, \ldots, x_\xi) = \prod_{\alpha=1}^{m} M_\alpha^{r_\alpha}(x_1, x_2, \ldots, x_\xi), \tag{14}$$

where each $M_\alpha^{r_\alpha}$ is a polynomial of the form (9) for the α^{th} distinct r-cycle and d_α is the multiplicity of that r-cycle; m is the number of cycle types in P_π. Linking back to the matrix formulation, each $M_\alpha^{r_\alpha}$ is equivalent to a *row* S_i in matrix S.

Since we know the fixed stoichiometry term $T = \prod_{i=1}^{\xi} T_i = \prod_{i=1}^{\xi} x_i^{c_i}$ *in advance, we can limit the possible sequences of* k_i *for which multinomial coefficients are calculated*. This is the key idea of the algorithm and the reason for its high performance.

For each group operation π, we have a product of polynomials $M_\alpha^{r_\alpha}$. We begin filtering the sequences by choosing only those combinations of values $v_{i\alpha} \in V_\alpha = \{v_{i\alpha}\}_{i=1}^{d_\alpha+1}$ for which the sum

$$\sum_{\alpha=1}^{m} v_{i\alpha} = T_i, \tag{15}$$

where V_α is the set from Eq. (10) for multinomial $M_\alpha^{r_\alpha}$. At this point it is useful to refer to Figure 3 to make the connection to the recursive tree search for possible matrices. The V_α are equivalent to all the possible values that any of the elements in a row of the matrix may take. If we take $M_1^{r_1}$ as an example, then V_1 is the collection of all values that show up in row 1 of any matrix in the figure, multiplied by the number of cycles with length r_1. Constraint (15) is equivalent to the column sum requirement (4).

We first apply constraint (15) to the x_1 term across the product of polynomials to find a set of values $\{k_{1\alpha}\}_{\alpha=1}^{m}$ that could give exponent T_1 once all the polynomials' terms have been expanded. This is equivalent to finding the set of first columns in each matrix that match the target frequency for the first color. Once a value $k_{1\alpha}$ has been fixed for each $M_\alpha^{r_\alpha}$, the remaining exponents in the sequence $\{k_{1\alpha}\} \cup \{k_{i\alpha}\}_{i=2}^{\xi}$ are constrained via (12). We can recursively examine each variable x_i in turn using these constraints to build a set of sequences

$$S_l = \{S_{l\alpha}\}_{\alpha=1}^{m} = \{(k_{1\alpha}, k_{2\alpha}, \ldots, k_{\xi\alpha})\}_{\alpha=1}^{m}, \tag{16}$$

where each $S_{l\alpha}$ defines the exponent sequence for its polynomial $M_\alpha^{r_\alpha}$ that will produce the target term T after the product is expanded. Each $S_{l\alpha} \in S_l$ represents the transposed matrix S that survives both the row and column sum conditions (highlighted in green in the figure). Thus, S_l is the set of these matrices for the group operation π. The

[4]We will use Greek subscripts to label the polynomials in the product and Latin subscripts to label the variables within any of the polynomials.

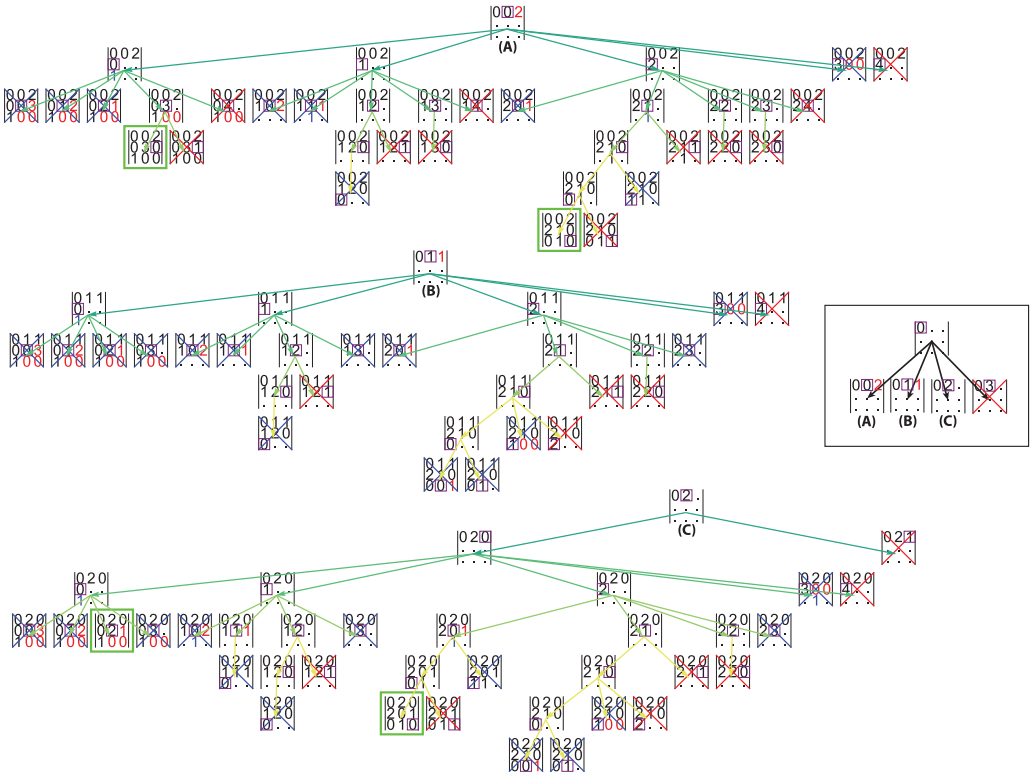

Fig. 3. A recursive tree search for some of the possible matrices S for the problem of Section 2: two 1-cycles, three 2-cycles, and one 4-cycle. We have restricted the figure to include only the zero pendants of the tree, which produce four of the five relevant matrices in Equation (6). Matrix elements in red (blue) represent the only possible values that would satisfy the row (column) sum conditions. A red (blue) cross over a matrix shows that it fails the row (column) sum condition, and its descendants need not be examined. Matrices with green borders are solutions to the tree search problem. The purple squares show the current row and column on which the recursive search is operating.

maximum value of l depends on the target term T and how many possible $v_{i\alpha}$ values are filtered out using constraints (15) and (12) at each step in the recursion.

Once the set $\mathbf{S} = \{S_l\}$ has been constructed, we use Equation (13) on each polynomial's $\{k_{i\alpha}\}_{i=1}^{\xi}$ in $S_{l\alpha}$ to find the contributing coefficients. The final coefficient value for term T resulting from group operation π is

$$t_\pi = \sum_l \tau_l = \sum_l \prod_{\alpha=1}^{m} \binom{d_\alpha}{S_{l\alpha}}. \tag{17}$$

To find the total number of unique colorings under the group action, this process is applied to each element $\pi \in \mathbf{G}$ and the results are summed and then divided by $|\mathbf{G}|$.

We can further optimize the search for contributing terms by ordering the exponents in the target term T in descending order. All the $\{k_{1\alpha}\}_{\alpha=1}^{m}$ need to sum to T_1 (15); larger values for T_1 are more likely to result in smaller sets of $\{k_{i\alpha}\}_{\alpha=1}^{m}$ across the polynomials. This happens because if T_1 has smaller values (like 1 or 2), then we end up with lots of possible ways to arrange them to sum to T_1 (which is not the the case for the larger values). Since the final set of sequences S_l is formed using a Cartesian product, including a few extra sequences from any T_i prunings multiplies the total number of

sequences significantly. In the figure, this optimization is equivalent to completing a row with red entries because all the remaining, unfilled entries are constrained by the row sum condition.

Additionally, constraint (12) applied within each polynomial will also reduce the total number of sequences to consider if the first variables x_1, x_2, and so on, are larger integers compared to the target values T_1, T_2, and so on. This speed-up comes from the recursive implementation: If x_1 is already too large (compared to T_1), then possible values for $x_2, x_3, \ldots$ are never considered. This optimization is equivalent to completing matrix columns with blue entries because of the column sum constraint.

3.1. Pseudocode Implementation

Note: Implementations in PYTHON and FORTRAN are available in the supplementary material.

For both algorithms presented below, the operator ($\Leftarrow$) pushes the value to its right onto the list to its left.

For algorithm (1) in the EXPAND procedure, the $\cup$ operator horizontally concatenates the integer *root* to an existing sequence of integers.

For BUILD_S$_l$, we use the exponent $k_{1\alpha}$ on the first variable in each polynomial to construct a full set of possible sequences for that polynomial. Those sets of sequences are then combined in SUM_SEQUENCES (alg. 2) using a Cartesian product over the sets in each multinomial.

When calculating multinomial coefficients, we use the form in Eq. (13) in terms of binomial coefficients with a fast, stable algorithm from Manolopoulos [2002].

In practice, many of the group operations π produce identical products $M_1^{r_1} M_2^{r_2} \ldots M_m^{r_m}$. Thus before computing any of the coefficients from the polynomials, we first form the polynomial products for each group operation and then add identical products together.

4. COMPUTATIONAL ORDER AND PERFORMANCE

The algorithm is structured around the *a priori* knowledge of the target stoichiometry. At the earliest possibility, we prune terms from individual polynomials that would not contribute to the final Pólya coefficient in the expanded product of polynomials (see Figure 3). Because the Pólya polynomial for each group operation is based on its disjoint-cyclic form, the complexity of the search can vary drastically from one group operation to the next. That said, it is common for groups to have several classes whose group operations (within each class) will have similar disjoint-cyclic forms and thus also scale similarly. However, from group to group, the set of classes and disjoint-cyclic forms may differ considerably; this makes it difficult to make a statement about the scaling of the algorithm in general. As such, we instead provide a formal, worst-case analysis for the algorithm's performance and supplement it with experimental examples. For these experiments, we crafted special groups with specific properties to demonstrate the various scaling behaviors as group properties change.

4.1. Worst-Case Scaling

Heuristically, the behavior of our algorithm should depend roughly on the size of the group: the number of permutations we have to analyze. That seems consistent with our experiments. But that can also be mitigated by noting that some groups of the same size have many more distinct cycle types than others. For example, if our group is generated by a single cycle of prime integer length p, then there are only two cycle types, despite the group having order p.

The majority of computation time should be spent in enumerating those matrices S and be proportional to the number of same (see Figure 4). Numerical experiments

ALGORITHM 1: Recursive Sequence Constructor

Procedure `initialize`(i, $k_{i\alpha}$, $M_\alpha^{r_\alpha}$, V_α,**T**)
Constructs a Sequence Object tree recursively for a single $M_\alpha^{r_\alpha}$ by filtering possible exponents on each x_i in the polynomial. The object has the following properties:
root: $k_{i\alpha}$, proposed exponent of x_i in $M_\alpha^{r_\alpha}$.
parent: proposed Sequence for $k_{i-1,\alpha}$ of x_{i-1}.
used: the sum of the proposed exponents to left of and including this variable $\sum_{j=1}^{i} k_{i\alpha}$.
i: index of variable in $M_\alpha^{r_\alpha}$ (column index).
$k_{i\alpha}$: proposed exponent of x_i in $M_\alpha^{r_\alpha}$ (matrix entry at $i\alpha$).
$M_\alpha^{r_\alpha}$: Pólya polynomial in P_π (14).
V_α: possible exponents for $M_\alpha^{r_\alpha}$ (10).
T: $\{T_i\}_{i=1}^{\xi}$ target stoichiometry.
. . . . . .
if $i = 1$ **then**
  self.used ← *self*.root + *self*.parent.used
else
  self.used ← *self*.root
end
self.kids ← empty
if $i \leq \xi$ **then**
  for $p \in V_\alpha$ **do**
    $rem \leftarrow p - self.$root
    if $0 \leq rem \leq T_i$ **and** $|rem| \leq d_\alpha r_\alpha - self.used$ **and** $|p - self.used| \bmod r_\alpha = 0$ **then**
      self.kids ⇐ `initialize`$(i+1, rem, M_\alpha^{r_\alpha}, V_\alpha, \mathbf{T})$
    end
  end
end

Function `expand`(*sequence*)
Generates a set of $S_{l\alpha}$ from a single Sequence object.
sequence: the object created using `initialize()`.
. . . . . .
sequences ← empty
for *kid* ∈ *sequence.kids* **do**
  for *seq* ∈ `expand`(*kid*) **do**
    sequences ⇐ *kid*.root ∪ *seq*
  end
end
if *len(sequence.kids)* $= 0$ **then**
  sequences ← {*kid*.root}
end
return *sequences*

Function `build_`S_l(**k**, **V**, P_π, **T**)
Constructs S_l from $\{k_{1\alpha}\}_{\alpha=1}^{m}$ for a P_π (14).
k: $\{k_{1\alpha}\}_{\alpha=1}^{m}$ set of possible exponent values on the *first* variable in each $M_\alpha^{r_\alpha} \in P_\pi$.
V: $\{V_\alpha\}_{\alpha=1}^{m}$ possible exponents for each $M_\alpha^{r_\alpha}$ (10).
P_π: Pólya polynomial representation for a single operation π in the group **G** (14).
T: $\{T_i\}_{i=1}^{\xi}$ target stoichiometry.
. . . . . .
sequences ← empty
for $\alpha \in \{1 \ldots m\}$ **do**
  seq ← `initialize`$(1, k_{1\alpha}, M_\alpha^{r_\alpha}, V_\alpha, \mathbf{T})$
  sequences ⇐ `expand`(*seq*)
end
return *sequences*

ALGORITHM 2: Coefficient Calculator

Function `sum_sequences`(S_l)
  Finds τ_l (17) *for* $S_l = \{S_{l\alpha}\}_{\alpha=1}^{m}$ (16)
  S_l: a set of lists (of exponent sequences $\{k_{i\alpha}\}_{i=1}^{\xi}$) for each polynomial $M_\alpha^{r_\alpha}$ in P_π (14).
  ...
  $K_l \leftarrow S_{l1} \times S_{l2} \times \cdots \times S_{lm} = \langle \{(k_{i\alpha})_{i=1}^{\xi}\}_{\alpha=1}^{m} \rangle_l$
  $coeff \leftarrow 0$
  for each $\{(k_{i\alpha})_{i=1}^{\xi}\}_{\alpha=1}^{m} \in K_l$ **do**
    if $\sum_{\alpha=1}^{m} k_{i\alpha} = T_i \; \forall \; i \in \{1 \ldots \xi\}$ **then**
      $coeff \leftarrow coeff + \prod_{\alpha=1}^{m} \binom{d_\alpha}{\{k_{i\alpha}\}_{i=1}^{\xi}}$
    end
  end
  return *coeff*
Function `coefficient`(**T**, P_π, **V**)
  Constructs $\mathbf{S} = \{S_l\}$ *and calculates* t_π (17)
  T: $\{T_i\}_{i=1}^{\xi}$ target stoichiometry.
  P_π: Pólya polynomial representation for a single operation π in the group **G** (14).
  V: $\{V_\alpha\}_{\alpha=1}^{m}$ possible exponents for each $M_\alpha^{r_\alpha}$ (10).
  ...
  if $m = 1$ **then**
    if $r_1 > T_i \; \forall \; i = 1..\xi$ **then**
      return 0
    else
      return $\binom{d_1}{T_1 T_2 \ldots T_\xi}$
    end
  else
    T $\leftarrow$ sorted(**T**)
    possible $\leftarrow V_1 \times V_2 \times \cdots \times V_m$
    coeffs $\leftarrow 0$
    for $\{k_{1\alpha}\}_{\alpha=1}^{m} \in$ *possible* **do**
      if $\sum_{\alpha=1}^{m} k_{1\alpha} = T_1$ **then**
        $S_l \leftarrow$ `build_`S_l`(`$\{k_{1\alpha}\}_{\alpha=1}^{m}$, **V**, P_π, **T**`)`
        coeffs $\leftarrow$ *coeffs*+ `sum_sequences(`S_l`)`
      end
    end
    return *coeffs*
  end

confirm[5] that the number of matrices scales exponentially with the number of colors (fixed group and number of elements in the set), linearly with the number of elements in the set (fixed number of colors and group), and is linear with the group size (fixed number of colors and elements in the set). The number of entries in the matrix S is $t\xi$ (see the discussion above Equation (3)) and the height of the entries is (roughly) bounded by the number of cycles and (very roughly) by the color frequencies divided by cycle lengths. This makes computing a time estimate based on these factors very difficult, but in the worst case, it could grow like the $t\xi$-th power of the average size of the entries, which will depend on the size of the target frequencies, and so on. This would be a very complex function to estimate, but we may expect it to grow exponentially for

[5]Figures are included in the code repository. See supplementary material.

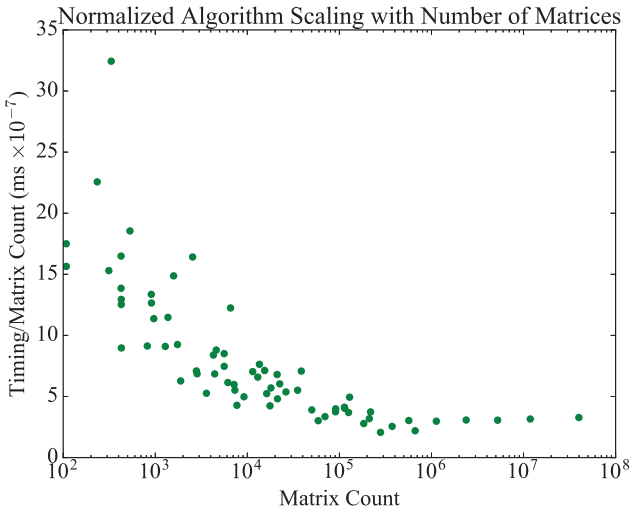

Fig. 4. Normalized algorithm scaling with the number of relevant matrices to enumerate. For large matrix counts, the behavior appears linear, supporting the hypothesis that the algorithm scales roughly with the number of matrices. The scatter is appreciable only for small matrix counts (less than 10^6).

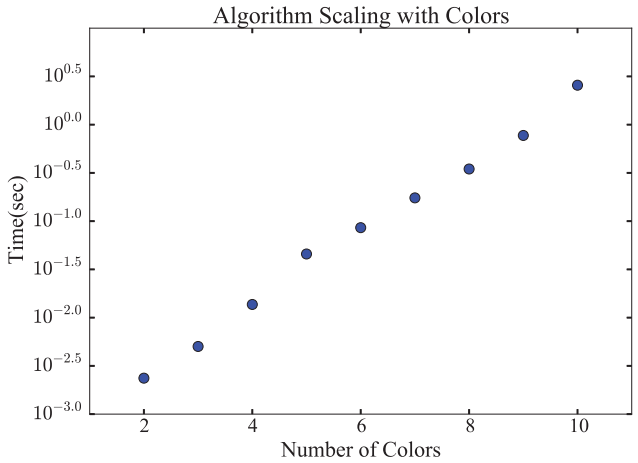

Fig. 5. Log plot of the algorithm scaling as the number of colors increases. Since the number of variables x_i in each polynomial increases with the number of colors, the combinatoric complexity of the expanded polynomial increases drastically with each additional color; this leads to an exponential scaling. The linear fit to the logarithmic data has a slope of 0.403.

very large input. We did not find that to be an impediment for the sizes of problems we needed to solve.

4.2. Experiments Demonstrating Algorithm Scaling

In Figure 5, we plot the algorithm's scaling as the number of colors in the enumeration increases (for a fixed group and number of elements). For each r-cycle in the disjoint-cyclic form of a group operation, we construct a polynomial with ξ variables, where ξ is the number of colors used in the enumeration. Because the group operation results in a product of these polynomials, increasing the number of colors by 1 increases the combinatoric complexity of the polynomial *expansion* exponentially. For

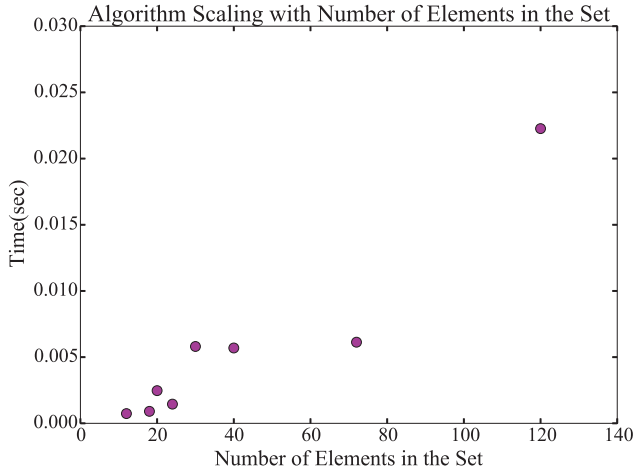

Fig. 6. Algorithm scaling as the number of elements in the finite set increases (for two colors). The Pólya polynomial arises from the group operations' disjoint-cyclic form, so more elements in the set results in a richer spectrum of possible polynomials multiplied together. Because of the algorithms aggressive pruning of terms, the exact disjoint-cyclic form of individual group operations has a large bearing on the algorithm's scaling. As such, it is not surprising that there is some scatter in the timings as the number of elements in the set increases.

this scaling experiment, we used the same transitive group acting on a finite set with 20 elements for each data point but increased the number of colors in the fixed color term T. We chose T by dividing the number of elements in the group as equally as possible; thus for two colors, we used $[10, 10]$; for three colors we used $[8, 6, 6]$, then $[5, 5, 5, 5]$, $[4, 4, 4, 4, 4]$, and so on. Figure 5 plots the $\log_{10}$ of the execution time (in ms) as the number of colors increases. As expected, the scaling is linear (on the log plot).

As the number of elements in the finite set increases, the possible Pólya polynomial representations for each group operation's disjoint-cyclic form increases exponentially. In the worst case, a group acting on a set with k elements may have an operation with k 1-cycles; on the other hand, that same group may have an operation with a single k-cycle, with lots of possibilities in between. Because of the richness of possibilities, it is almost impossible to make general statements about the algorithm's scaling without knowing the structure of the group and its classes. In Figure 6, we plot the scaling for a set of related groups (all are isomorphic to the direct product of $S_3 \times S_4$) applied to finite sets of varying sizes. Every data point was generated using a transitive group with 144 elements. Thus, this plot shows the algorithm's scaling when the group is the same and the number of elements in the finite set changes. Although the scaling appears almost linear, there is a lot of scatter in the data. Given the rich spectrum of possible Pólya polynomials that we can form as the set size increases, the scatter is not surprising.

Finally, we consider the scaling as the group size increases (Figure 7). For this test, we selected the set of unique groups arising from the enumeration of all derivative super structures of a simple cubic lattice for a given number of sites in the unit cell [Hart and Forcade 2008]. Since the groups are formed from the symmetries of real crystals, they arise from the semidirect product of operations related to physical rotations and translations of the crystal. In this respect, they have similar structure for comparison. In most cases, the scaling is obviously linear; however, the slope of each trend varies from group to group. This once again highlights the scaling's heavy dependence on

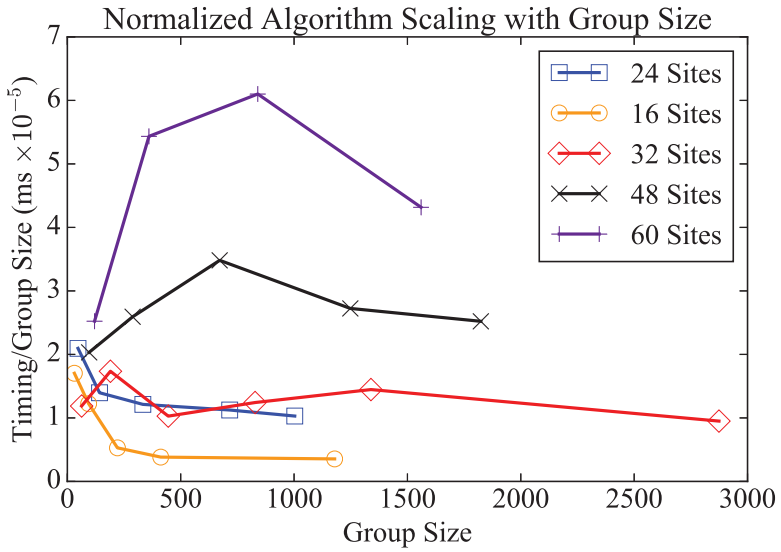

Fig. 7. Normalized algorithm scaling with group size for an enumeration problem from solid state physics [Hart and Forcade 2008]. We used the unique permutation groups arising from all derivative super structures of a simple cubic lattice for a given number of sites in the unit cell. The behavior is generally linear with increasing group size.

the specific disjoint-cyclic forms of the group operations. Even for groups with obvious similarity, the scaling may differ.

4.3. Comparison with Computer Algebra Systems

In addition to the explicit timing analysis and experiments presented above, we also ran a group of representative problems with our algorithm and MATHEMATICA (a common CAS). We also attempted the tests with MAPLE but were unable to obtain consistent results between multiple runs of the same problems.[6] So, we have opted to exclude the MAPLE timing results. For the comparison with MATHEMATICA, we used MATHEMATICA's `Expand` and `Coefficient` functions to return the relevant coefficient from the Pólya polynomial (see Figure 8).

5. SUMMARY

Until now, no low-level, numerical implementation of Pólya's enumeration theorem has been readily available; instead, a CAS was used to symbolically solve the polynomial expansion problem posed by Pólya. While CAS's are effective for smaller, simpler calculations, as the difficulty of the problem increases, they become impractical solutions. Additionally, codes that perform the actual enumeration of the colorings are often implemented in low-level codes, and interoperability with a CAS is not necessarily easy to automate.

We presented a low-level, purely numerical algorithm and code that exploits the properties of polynomials to restrict the combinatoric complexity of the expansion. By considering only those coefficients in the unexpanded polynomials that might contribute to the final answer, the algorithm reduces the number of terms that must be included to find the significant term in the expansion.

[6]The inconsistency manifests in MAPLE sometimes returning 0 instead of the correct result and sometimes running the same problem unpredictably in hours or seconds.

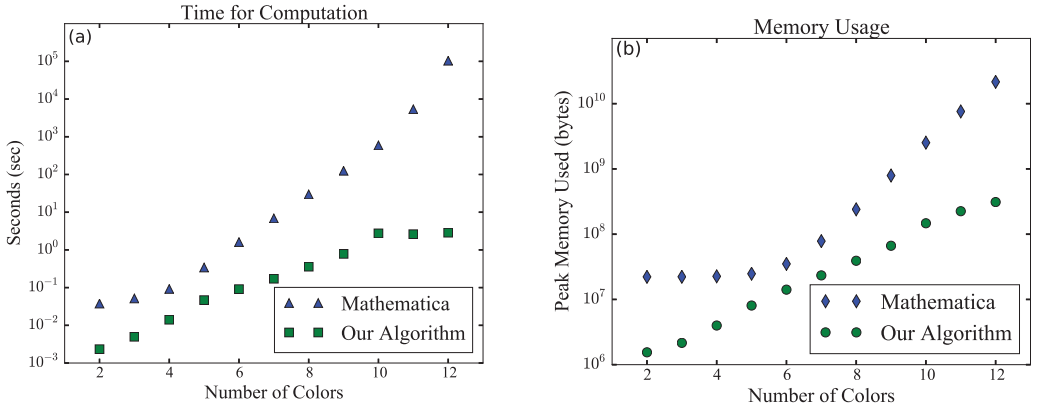

Fig. 8. Comparison of the CPU time (a) and memory usage (b) between the FORTRAN implementation of our algorithm and MATHEMATICA as the number of colors increases. These are the times needed to generate the data in Figure 5.

Because of the algorithm scaling's reliance on the exact structure of the group and the disjoint-cyclic form of its operations, a rigorous analysis of the scaling is not possible without knowledge of the group. Instead, we presented some numerical timing results from representative, real-life problems that show the general scaling behavior.

In contrast to the CAS solutions whose execution times range from milliseconds to hours, our algorithm consistently performs in the millisecond to second regime, even for complex problems. Additionally, it is already implemented in both high- and low-level languages, making it useful for confirming enumeration results. This makes it an effective substitute for alternative CAS implementations.

REFERENCES

Stefano Curtarolo, Gus L. W. Hart, Marco Buongiorno Nardelli, Natalio Mingo, Stefano Sanvito, and Ohad Levy. 2013. The high-throughput highway to computational materials design. *Nat. Mater.* 12, 3 (MAR 2013), 191–201. DOI:http://dx.doi.org/10.1038/NMAT3568

Kecai Deng and Jianguo Qian. 2014. Enumerating stereo-isomers of tree-like polyinositols. *J. Math. Chem.* 52, 6 (2014), 1581–1598.

Roberto Dovesi, Roberto Orlando, Alessandro Erba, Claudio M. Zicovich-Wilson, Bartolomeo Civalleri, Silvia Casassa, Lorenzo Maschio, Matteo Ferrabone, Marco De La Pierre, Philippe D'Arco, Yves Nol, Mauro Caus, Michel Rrat, and Bernard Kirtman. 2014. CRYSTAL14: A program for the ab initio investigation of crystalline solids. *Int. J. Quant. Chem.* 114, 19 (2014), 1287–1317. DOI:http://dx.doi.org/10.1002/qua.24658

Antoine Genitrini, Bernhard Gittenberger, Veronika Kraus, and Cécile Mailler. 2015. Associative and commutative tree representations for Boolean functions. *Theor. Comput. Sci.* 570 (2015), 70–101.

Modjtaba Ghorbani and Mahin Songhori. 2014. The enumeration of Chiral isomers of tetraammine platinum (II). *Match-Communications in Mathematical and in Computer Chemistry* 71, 2 (2014), 333–340.

Frank Harary. 1955. The number of linear, directed, rooted, and connected graphs. *Trans. Am. Math. Soc.* 78, 2 (1955), 445–463.

Gus L. W. Hart and Rodney W. Forcade. 2008. Algorithm for generating derivative structures. *Phys. Rev. B* 77 (Jun 2008), 224115. Issue 22. DOI:http://dx.doi.org/10.1103/PhysRevB.77.224115

Gus L. W. Hart and Rodney W. Forcade. 2009. Generating derivative structures from multilattices: Application to HCP alloys. *Phys. Rev. B* 80 (July 2009), 014120.

Gus L. W. Hart, Lance J. Nelson, and Rodney W. Forcade. 2012. Generating derivative structures for a fixed concentration. *Comp. Mat. Sci.* 59 (2012), 101–107. DOI:http://dx.doi.org/10.1016/j.commatsci.2012.02.015

B. A. Kennedy, D. A. McQuarrie, and C. H. Brubaker Jr. 1964. Group theory and isomerism. *Inorg. Chem.* 3, 2 (1964), 265–268.

Peter Lackner, Harald Fripertinger, and Gerhard Nierhaus. 2015. Peter Lackner/tropical investigations. In *Patterns of Intuition*. Springer, Berlin, 279–313.

Yannis Manolopoulos. 2002. Binomial coefficient computation: Recursion or iteration? *ACM SIGCSE Bulletin InRoads* 34 (Dec 2002). Issue 4. DOI:http://dx.doi.org/10.1145/820127.820168

James McGrane, Sanjaye Ramgoolam, and Brian Wecht. 2015. Chiral ring generating functions & branches of moduli space. *arXiv preprint arXiv:1507.08488* (2015).

Sami Mustapha, Philippe DArco, Marco De La Pierre, Yves Nol, Matteo Ferrabone, and Roberto Dovesi. 2013. On the use of symmetry in configurational analysis for the simulation of disordered solids. *J. Phys.: Condens. Matter* 25, 10 (2013), 105401. http://stacks.iop.org/0953-8984/25/i=10/a=105401.

George Pólya. 1937. Kombinatorische anzahlbestimmungen fr gruppen, graphen und chemische verbindungen. *Acta Math.* 68, 1 (1937), 145–254.

George Pólya and Ronald C. Read. 1987. *Combinatorial Enumeration of Groups, Graphs, and Chemical Compounds* (1987).

Jianguo Qian. 2014. Enumeration of unlabeled uniform hypergraphs. *Discr. Math.* 326, 1 (2014), 66–74.

R. W. Robinson, F. Harry, and A. T. Balaban. 1976. The numbers of chiral and achiral alkanes and monosubstituted alkanes. *Tetrahedron* 32, 3 (1976), 355–361.

Masahiko Taniguchi, Sarah Henry, Richard J. Cogdell, and Jonathan S. Lindsey. 2014. Statistical considerations on the formation of circular photosynthetic light-harvesting complexes from rhodopseudomonas palustris. *Photosynth. Res.* 121, 1 (2014), 49–60.

J. Tura, R. Augusiak, A. B. Sainz, B. Lücke, C. Klempt, M. Lewenstein, and A. Acín. 2015. Nonlocality in many-body quantum systems detected with two-body correlators. *arXiv preprint arXiv:1505.06740* (2015).

Received December 2015; revised May 2016; accepted June 2016