Noname manuscript No.
(will be inserted by the editor)

Numerical Algorithm for Pólya Enumeration Theorem

Conrad W. Rosenbrock · Wiley S. Morgan · Gus L. W. Hart · Stefano Curtarolo · Rodney W. Forcade

Received: date / Accepted: date

Abstract 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.

Keywords Pólya enumeration theorem · expansion coefficient · product of polynomials

1 Introduction

A common problem in many fields involves enumerating the possible colorings of a finite set. Applying a symmetry or permutation group reduces the size of the enumerated set by including only those elements that are unique under the group action. The Pólya enumeration theorem counts the number of unique colorings that should be recovered [10]. The Pólya theorem has shown its wide range of applications in a variety of contexts, such as confirming enumerations of molecules in bioinformatics and chemoinformatics [2]; unlabeled, uniform hypergraphs in discrete mathematics [11]; and photosensitisers in photosynthesis research [12].

Conrad W. Rosenbrock · Wiley S. Morgan · Gus L. W. Hart
Department of Physics and Astronomy, Brigham Young University, Provo, UT, 84602
Tel.: +1-801-422-7444
E-mail: gus.hart@gmail.com

Rodney W. Forcade
Department of Mathematics, Brigham Young University, Provo, UT, 84602, USA

Stefano Curtarolo
Materials Science, Electrical Engineering, Physics and Chemistry, Duke University, Durham, NC 27708, USA

Typical implementations of the counting theorem use Computer Algebra Systems to symbolically solve the polynomial coefficient problem. However, despite the widespread use of the theorem, a low-level numerical implementation for recovering the number of unique colorings is not readily available. Although a brute-force calculation of the expansion coefficients for the Pólya polynomial is straight-forward to implement, it is prohibitively slow. For instance, we recently used such a brute force method to confirm enumeration results for a lattice coloring problem in solid state physics [4]. After profiling performance on more than 20 representative systems, we found that the brute force calculation of the Pólya coefficient took as long as the enumeration problem itself. Here we demonstrate that the performance can be improved drastically by exploiting the properties of polynomials. The improved performance also enables harder Pólya theorem problems to be easily solved that would otherwise be computationally prohibitive [1].

We first 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 both heuristically and via numerical experimentation.

2 Pólya Enumeration Theorem

Because of extensive literature coverage, we do not derive the Pólya's theorem here[2]. Rather, we just state its main claims by using a simple example.

[1] For example, in one test we performed, Mathematica required close to 5 hours to compute the coefficient, while our algorithm found the same answer in 0.2 seconds.

[2] The interested reader may refers to Refs. [9, 10]

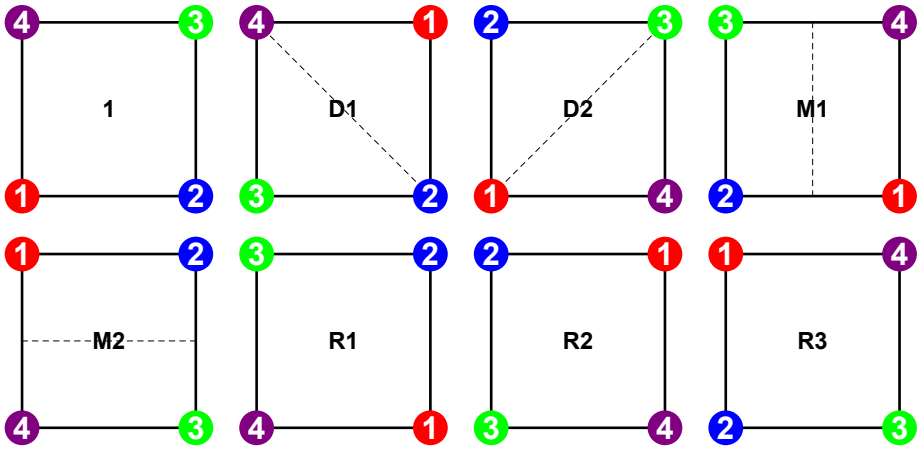

Fig. 1 The symmetry group operations of the square. This group is known as the dihedral group of degree 4, or D_4. The dashed lines are guides to the eye for the horizontal, vertical and diagonal reflections (**M1**,**M2** and **D1**, **D2**).

The square has the set of symmetries displayed in Figure 1. These symmetries include three rotations (by 90, 180 and 270 degrees; labelled **R1**, **R2**, and **R3**) and four reflections (one horizontal, one vertical and two for the diagonals; labelled **M1**, **M2** and **D1**, **D2**). This group is commonly known as the dihedral group of degree four, or D_4 for short[3].

The group operations of the D_4 group can be written in disjoint-cyclic form as in Table 1. For each r-cycle in the group, we can write a polynomial in variables x_i^r for $i = 1 \dots \xi$, 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 just 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 that it is part of. For a general r-cycle, the polynomial takes the form

$$(x_1^r + x_2^r + \dots + x_\xi^r), \tag{1}$$

for an enumeration with ξ colors. Most group operations will have a product of these polynomials for each r-cycle in the disjoint-cyclic form. Once the product of polynomials has been generated with the group operation, we can simplify it by adding exponents to identical polynomials. In the example above, $(x+y)(x+y)$ would become $(x+y)^2$; in summary, 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 will count how many *unique* colorings we should recover. To find out the expected number of unique colorings, 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 1. 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 $(6+2+2+2+0+2+0)/8 = 16/8 = 2$. The unique colorings are plotted in Figure 2.

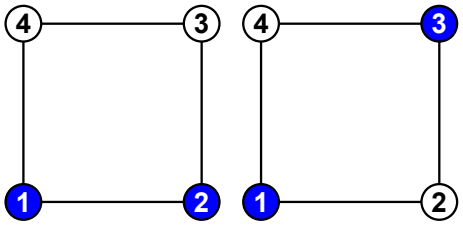

Fig. 2 The two unique ways to color the square with two colors and two corners of each color.

Generally, for a finite set with F elements, and fixed color concentrations c_i such that $\sum_{i=1}^{\xi} c_i = F$, the number of unique colorings of the set under the group action corresponds to the coefficient of the term

$$T = x_1^{c_1} x_2^{c_2} \dots x_\xi^{c_\xi} = \prod_i^{\xi} x_i^{c_i} \tag{2}$$

in the expanded polynomial for each group operation, summed over all elements in the group. Counting the number of unique colorings at fixed concentration amounts to finding the coefficient of a specific term, known *a priori*, from a product of polynomials.

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

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

Table 1 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.

3 Coefficient-Finding Algorithm

We begin by reviewing some well-known properties of polynomials with respect to their variables. First, for a generic polynomial

$$(x_1^r + x_2^r + \cdots + x_\xi^r)^d, \tag{3}$$

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

$$V = \{0, r, 2r, 3r, \ldots, dr\}. \tag{4}$$

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

$$(x_1^r + x_2^r + \cdots + x_\xi^r)^d = \sum_{k_1,k_2,\ldots,k_\xi} \mu_k \prod_{i=1}^{\xi} x_i^{rk_i} \tag{5}$$

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,

$$k_1 + k_2 + \cdots + k_\xi = d. \tag{6}$$

The coefficients μ_k in the polynomial expansion Equation (5) are found using the multinomial tcoefficients

$$\begin{aligned}\mu_k &= \binom{n}{k_1, k_2, \ldots, k_\xi} = \frac{n!}{k_1!k_2!\cdots k_\xi!} \\ &= \binom{k_1}{k_1}\binom{k_1+k_2}{k_2}\cdots\binom{k_1+k_2+\cdots+k_\xi}{k_\xi} \\ &= \prod_{i=1}^{\xi}\binom{\sum_{j=1}^{i} k_j}{k_i}.\end{aligned} \tag{7}$$

Finally, we define the polynomial (1) for an arbitrary group operation $G^i \in \mathbf{G}$ as[4]

$$P^i(x_1, x_2, \ldots, x_\xi) = \prod_{\alpha=1}^{m} M_\alpha^{r_\alpha}(x_1, x_2, \ldots, x_\xi) \tag{8}$$

[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.

where each $M_\alpha^{r_\alpha}$ is a polynomial for the α^{th} distinct r-cycle of the form (3) and d_α is substituted for the value of d (which is the multiplicity of that r-cycle); m is the number of distinct values of r_α in P^i.

Since we know the fixed concentration term $T = \prod_{i=1}^{\xi} T_i = \prod_{i=1}^{\xi} x_i^{c_i}$ *in advance (see equation (2)), 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 G^i, 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

$$\sum_{\alpha=1}^{m} v_{i\alpha} = T_i \tag{9}$$

where V_α is the set from eqn. (4) for multinomial $M_\alpha^{r_\alpha}$.

We first apply constraint (9) 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. 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 (6). We can recursively examine each variable x_i in turn using these constraints to build a set of sequences

$$S_l = \{S_{l\alpha}\}_{\alpha=1}^{m} = \{(k_{1\alpha}, k_{2\alpha}, \ldots, k_{\xi\alpha})\}_{\alpha=1}^{m} \tag{10}$$

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. The maximum value of l depends on the target term T and how many possible $v_{i\alpha}$ values are filtered out using constraints (9) and (6) at each step in the recursion.

Once the set $\mathbf{S} = \{S_l\}$ has been constructed, we use Equation (7) 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 operation G^i is

$$t_i = \sum_l \tau_l = \sum_l \prod_{\alpha=1}^{m} \binom{d_\alpha}{S_{l\alpha}}. \tag{11}$$

To find the total number of unique colorings under the group action, this process is applied to each element $G^i \in \mathbf{G}$ 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. Because the possible sequences $\{k_{1\alpha}\}_{\alpha=1}^m$ are filtered using T_1, larger values for T_1 are more likely to result in smaller sets of $\{k_{i\alpha}\}_{\alpha=1}^m$ across the polynomials. All the $\{k_{1\alpha}\}_{\alpha=1}^m$ need to sum to T_1 (9); if T_1 has smaller values (like 1 or 2), we will 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, having a few extra sequences from the T_1 pruning multiplies the total number of sequences significantly. Additionally, constraint (6) applied within each polynomial will also reduce the total number of sequences to consider if the first variables x_1, x_2, etc. are larger integers.

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.

For algorithm (2) in the SUM_SEQUENCES function, K_l is calculated using the cartesian product of the individual $S_{l\alpha}$, where for a given l, the number of sequences $\{k_i\}_{i=1}^{\xi} \in S_{l\alpha}$ may be arbitrary. For example, a product of three polynomials $M_1^4 M_2^3 M_3^2$ may produce possible sequences with $|S_{l1}| = 2$, $|S_{l2}| = 4$ and $|S_{l3}| = 4$. Then $|K_l| = 2 \times 4 \times 4 = 32$ and each element in K_l is a set of three sequences: $\{(k_{i\alpha})_{i=1}^{\xi}\}_{\alpha=1}^3$, one for each polynomial, which specifies the exponents on the contributing term from that polynomial. Also, when calculating multinomial coefficients, we use the form in eqn. (7) in terms of binomial coefficients with a fast, stable algorithm from Manolopoulos [7].

In practice, many of the group operations G^i produce identical products $M_1^{r_1} M_2^{r_2} \dots M_m^{r_m}$. Thus before computing any of the coefficients from the polynomials,

Algorithm 1 Recursive Sequence Constructor

procedure INITIALIZE(i, $k_{i\alpha}$, $M_\alpha^{r_\alpha}$, V_α, $\mathbf{T}$)
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 variable x_i in $M_\alpha^{r_\alpha}$.
parent: proposed Sequence object 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 $M_\alpha^{r_\alpha}$
$k_{i\alpha}$: proposed exponent of variable x_i in $M_\alpha^{r_\alpha}$.
$M_\alpha^{r_\alpha}$: Pólya polynomial representation of a single polynomial in P^i (8).
V_α: possible exponents for $M_\alpha^{r_\alpha}$ (4).
$\mathbf{T}$: $\{T_i\}_{i=1}^{\xi}$ exponents for the concentration term (2).
. . . . . .
if $i = 1$ **then**
self.used $\leftarrow$ *self*.root + *self*.parent.used
else
self.used $\leftarrow$ *self*.root
self.kids $\leftarrow$ 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 $\Leftarrow$ Sequence$(i+1, rem, M_\alpha^{r_\alpha}, V_\alpha, \mathbf{T})$

function EXPAND(sequence)
Generates a set of $S_{l\alpha}$ from a single Sequence object.
sequence: the object created using INITIALIZE.
. . . . . .
sequences $\leftarrow$ empty
for *kid* $\in$ sequence.kids **do**
for *seq* $\in$ EXPAND(*kid*) **do**
sequences $\Leftarrow$ *kid*.root $\cup$ *seq*
if len(sequence.kids) $= 0$ **then**
sequences $\leftarrow$ {*kid*.root}
return *sequences*

function BUILD_S$_l$($\mathbf{k}$, $\mathbf{V}$, P^i, $\mathbf{T}$)
Constructs S_l from $\{k_{1\alpha}\}_{\alpha=1}^m$ for a P^i (8).
$\mathbf{k}$: $\{k_{1\alpha}\}_{\alpha=1}^m$ set of possible exponent values on the *first* variable in each $M_\alpha^{r_\alpha} \in P^i$.
$\mathbf{V}$: $\{V_\alpha\}_{\alpha=1}^m$ possible exponents for each $M_\alpha^{r_\alpha}$ (4).
P^i: Pólya polynomial representation for a single operation in the group $\mathbf{G}$ (8).
$\mathbf{T}$: $\{T_i\}_{i=1}^{\xi}$ exponents for the concentration term (2).
. . . . . .
sequences $\leftarrow$ empty
for $\alpha \in \{1 \dots m\}$ **do**
seq $\leftarrow$ INITIALIZE$(1, k_{1\alpha}, M_\alpha^{r_\alpha}, V_\alpha, \mathbf{T})$
sequences $\Leftarrow$ EXPAND(*seq*)
return *sequences*

Algorithm 2 Coefficient Calculator

function SUM_SEQUENCES(S_l)
  Finds τ_l (11) *for* $S_l = \{S_{l\alpha}\}_{\alpha=1}^{m}$ (10)
  S_l: a set of lists (of exponent sequences $\{k_{i\alpha}\}_{i=1}^{\xi}$) for each polynomial $M_\alpha^{r_\alpha}$ in the product P^i (8).
  .....
  $K_l \leftarrow S_{l1} \times S_{l2} \times \cdots \times S_{lm} = \langle \{(k_{i\alpha})_{i=1}^{\xi}\}_{\alpha=1}^{m} \rangle_l$
  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* $+ \prod_{\alpha=1}^{m} \binom{d_\alpha}{\{k_{i\alpha}\}_{i=1}^{\xi}}$
  return *coeff*

function COEFFICIENT($\mathbf{T}$, P^i, $\mathbf{V}$)
  Constructs $\mathbf{S} = \{S_l\}$ *and calculates* t_i (11)
  $\mathbf{T}$: $\{T_i\}_{i=1}^{\xi}$ exponents for the concentration term (2).
  P^i: Pólya polynomial representation for a single operation in the group $\mathbf{G}$ (8).
  $\mathbf{V}$: $\{V_\alpha\}_{\alpha=1}^{m}$ possible exponents for each $M_\alpha^{r_\alpha}$ (4).
  .....
  if $m = 1$ **then**
    if $r_1 > T_i \; \forall \; i = 1..\xi$ **then**
      return 0
    else
      return $\binom{d_1}{T_1 T_2 \ldots T_\xi}$
  else
    $\mathbf{T} \leftarrow$ sorted($\mathbf{T}$)
    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_$S_l(\{k_{1\alpha}\}_{\alpha=1}^{m}, \mathbf{V}, P^i, \mathbf{T})$
        coeffs $\leftarrow$ *coeffs* $+$ SUM_SEQUENCES(S_l)
    return *coeffs*

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 fixed concentration term (2). At the earliest possibility, we prune terms from individual polynomials that would not contribute to the final polya coefficient in the expanded product of polynomials. 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 be very different; this makes it difficult to make a statement about the scaling of the algorithm in general. Although we could make statements about the scaling of well-known sets of groups (for example the dihedral groups used in our example above), we decided instead to craft certain special groups with specific properties and run tests to determine the scaling numerically.

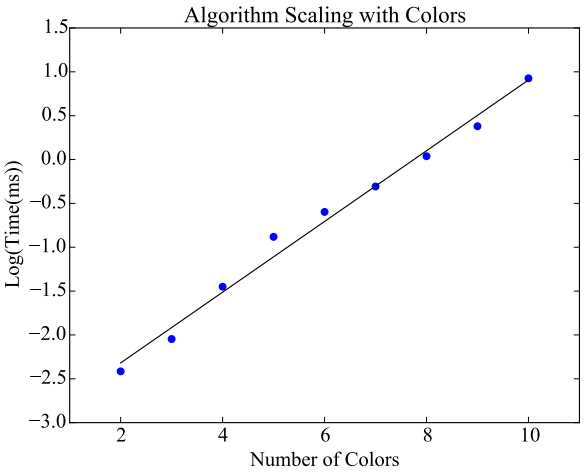

Fig. 3 Log plot of the algorithm scaling as the number of colors increases. Since the number of variables x_i in each polynomial increases with the number of colors, the combinatoric complexity of the expanded polynomial increases drastically with each additional color; this leads to an exponential scaling. The linear fit to the logarithmic data has a slope of 0.403.

In Figure 3 we plot the algorithm's scaling as the number of colors in the enumeration increases. 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 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 2 colors, we used $[10, 10]$; for 3 colors we used $[8, 6, 6]$, then $[5, 5, 5, 5]$, $[4, 4, 4, 4, 4]$, etc. Figure 3 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). The linear fit to the data points has a slope of 0.403.

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

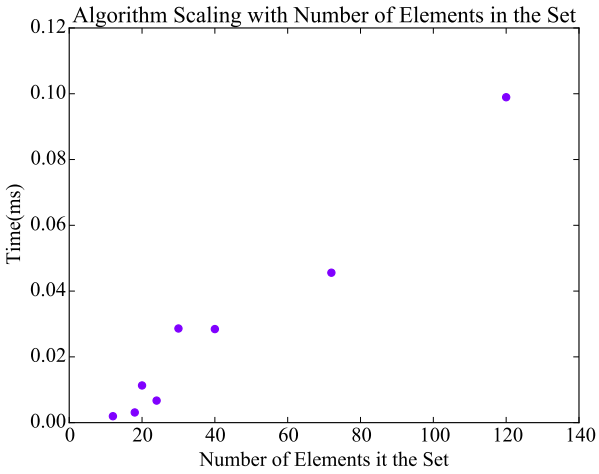

Fig. 4 Algorithm scaling as the number of elements in the finite set increases. The Pólya polynomial arises from the group operations' disjoint-cyclic form, so that more elements in the set results in a richer spectrum of possible polynomials multiplied together. Because of the algorithms aggresive pruning of terms, the exact disjoint-cyclic form of individual group operations has a large bearing on the algorithm's scaling. As such it isn't surprising that there is some scatter in the timings as the number of elements in the set increases.

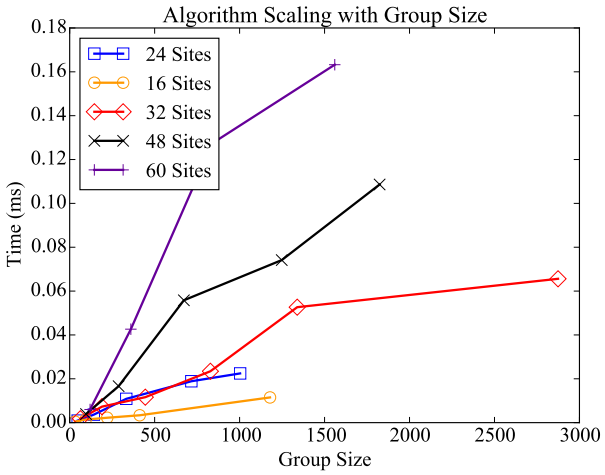

Fig. 5 Algorithm scaling with group size for an enumeration problem from solid state physics [4]. We used the unique permutation groups arising from all derivative super structures of a simple cubic lattice for a given number of sites in the unit cell. The behavior is generally linear with increasing group size.

is almost impossible to make general statements about the algorithm's scaling without knowing the structure of the group and its classes. In Figure 4, 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 isn't surprising.

Finally, we consider the scaling as the group size increases. 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 [4]. Since the groups are formed from the symmetries of real crystals, they arise from the direct 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 the specific disjoint-cyclic forms of the group operations. Even for groups with obvious similarity, the scaling may be different.

5 Summary

Until now, no low-level, numerical implementation of Pólya's enumeration theorem was readily available; instead, a computer algebra system (CAS) was used to symbolically solve the polynomial expansion problem posed by Pólya. While such systems are effective for small, 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 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.

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. Because all the timings are in the millisecond to second regime anyway, a more rigorous analysis of the algorithm's scaling is unnecessary.

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 easy to implement in low-level languages, making it useful for confirming enumeration results. This makes it an effective substitute for alternative CAS implementations.

In computational materials science, chemistry, and related subfields such as computational drug discovery, combinatorial searches are becoming increasingly important, especially in high-throughput studies [1]. The upside potential of these efforts continues to grow because computing power continues to become cheaper and algorithms continue to evolve. As computational methods gain a larger market share in materials discovery, algorithms such as this one are important as they provide validation support to complex simulation codes. The present algorithm has been useful in checking a new algorithm extending the work in Refs. [4–6], and Pólya's theorem was recently used in Mustapha's enumeration algorithm[8] that has been incorporated into the CRYSTAL14 software package [3].

Acknowledgements This work was supported under ONR (MURI N00014-13-1-0635).

References

1. Curtarolo, S., Hart, G.L.W., Nardelli, M.B., Mingo, N., Sanvito, S., Levy, O.: The high-throughput highway to computational materials design. Nature Materials **12**(3), 191–201 (2013). DOI 10.1038/NMAT3568
2. Deng, K., Qian, J.: Enumerating stereo-isomers of tree-like polyinositols. Journal of Mathematical Chemistry **52**(6), 1581–1598 (2014)
3. Dovesi, R., Orlando, R., Erba, A., Zicovich-Wilson, C.M., Civalleri, B., Casassa, S., Maschio, L., Ferrabone, M., De La Pierre, M., D'Arco, P., Nol, Y., Caus, M., Rrat, M., Kirtman, B.: Crystal14: A program for the ab initio investigation of crystalline solids. International Journal of Quantum Chemistry **114**(19), 1287–1317 (2014). DOI 10.1002/qua.24658. URL http://dx.doi.org/10.1002/qua.24658
4. Hart, G.L.W., Forcade, R.W.: Algorithm for generating derivative structures. Phys. Rev. B **77**, 224,115 (2008). DOI 10.1103/PhysRevB.77.224115. URL http://link.aps.org/doi/10.1103/PhysRevB.77.224115
5. Hart, G.L.W., Forcade, R.W.: Generating derivative structures from multilattices: Application to hcp alloys. Phys. Rev. B **80**, 014,120 (2009)
6. Hart, G.L.W., Nelson, L.J., Forcade, R.W.: Generating derivative structures for a fixed concentration. Comp. Mat. Sci. **59**, 101–107 (2012). DOI 10.1016/j.commatsci.2012.02.015
7. Manolopoulos, Y.: Binomial coefficient computation: Recursion or iteration? ACM SIGCSE Bulletin InRoads **34** (2002). DOI 10.1145/820127.820168. URL http://delab.csd.auth.gr/papers/SBI02m.pdf
8. Mustapha, S., DArco, P., Pierre, M.D.L., Nol, Y., Ferrabone, M., Dovesi, R.: On the use of symmetry in configurational analysis for the simulation of disordered solids. Journal of Physics: Condensed Matter **25**(10), 105,401 (2013). URL http://stacks.iop.org/0953-8984/25/i=10/a=105401
9. Pólya, G.: Kombinatorische anzahlbestimmungen fr gruppen, graphen und chemische verbindungen. Acta Mathematica **68**(1), 145–254 (1937)
10. Pólya, G., Read, R.C.: Combinatorial Enumeration of Groups, Graphs, and Chemical Compounds (1987)
11. Qian, J.: Enumeration of unlabeled uniform hypergraphs. Discrete Mathematics **326**(1), 66–74 (2014)
12. Taniguchi, M., Henry, S., Cogdell, R.J., Lindsey, J.S.: Statistical considerations on the formation of circular photosynthetic light-harvesting complexes from rhodopseudomonas palustris. Photosynthesis Research **121**(1), 49–60 (2014)

6 Supplementary Material

The source code to implement this algorithm is available for both python and Fortran at:

https://github.com/rosenbrockc/polya

The home page on github has full instructions for using either version of the code as well a battery of over 50 unit tests that were used to verify and time the algorithm. The unit tests can be executed using the FORTPY framework available via the Python Package Index. Instructions for running the unit tests are also on the github home page.