Brigham Young University
BYU ScholarsArchive

All Theses and Dissertations

2016-09-01

Increasing the Computational Efficiency of Combinatoric Searches

Wiley Spencer Morgan
Brigham Young University

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Astrophysics and Astronomy Commons

BYU ScholarsArchive Citation

Morgan, Wiley Spencer, "Increasing the Computational Efficiency of Combinatoric Searches" (2016). *All Theses and Dissertations*. 6528.
https://scholarsarchive.byu.edu/etd/6528

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.

Increasing the Computational Efficiency of Combinatoric Searches

Wiley Spencer Morgan

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Gus L. W. Hart, Chair
Branton Campbell
John Colton

Department of Physics and Astronomy

Brigham Young University

Copyright © 2016 Wiley Spencer Morgan

All Rights Reserved

ABSTRACT

Increasing the Computational Efficiency of Combinatoric Searches

Wiley Spencer Morgan
Department of Physics and Astronomy, BYU
Master of Science

A new algorithm for the enumeration of derivative superstructures of a crystal is presented. The algorithm will help increase the efficiency of computational material design methods such as the cluster expansion by increasing the size and diversity of the types of systems that can be modeled. Modeling potential alloys requires the exploration of all possible configurations of atoms. Additionally, modeling the thermal properties of materials requires knowledge of the possible ways of displacing the atoms. One solution to finding all symmetrically unique configurations and displacements is to generate the complete list of possible configurations and remove those that are symmetrically equivalent. This approach, however, suffers from the combinatoric explosion that happens when the supercell size is large, when there are more than two atom types, or when atomic displacements are included in the system. The combinatoric explosion is a problem because the large number of possible arrangements makes finding the relatively small number of unique arrangements for these systems impractical. The algorithm presented here is an extension of an existing algorithm [1–3] to include the extra configurational degree of freedom from the inclusion of displacement directions. The algorithm makes use of another recently developed algorithm for the Pólya [4–6] counting theorem to inform the user of the total number of unique arrangements before performing the enumeration and to ensure that the list of unique arrangements will fit in system memory. The algorithm also uses group theory to eliminate large classes of arrangements rather than eliminating arrangements one by one. The three major topics of this paper will be presented in this order, first the Pòlya algorithm, second the new algorithm for eliminating duplicate structures, and third the algorithms extension to include displacement directions. With these tools, it is possible to avoid the combinatoric explosion and enumerate previously inaccessible systems, including those that contain displaced atoms.

Keywords: enumeration, algorithm, combinatorics

ACKNOWLEDGMENTS

I would like to thank Conrad Rosenbrock for his aid in writing efficient algorithms, Dr. Forcade for the use of his extensive knowledge of group theory, and Dr. Hart for his guidance and assistance throughout this project.

Contents

Table of Contents **iv**

List of Figures **vi**

List of Tables **ix**

1 Introduction **1**

1.1 Background 1

1.2 Methods of Generating Derivative Superstructures 5

2 Pòlya's Enumeration Algorithm **7**

2.1 Introduction 8

2.2 Pólya Enumeration Theorem 10

2.2.1 Introduction to the Pólya Enumeration Theorem 10

2.2.2 Example: Applying Pólyas Theorem 14

2.3 Coefficient-Finding Algorithm 17

2.3.1 Pseudocode Implementation 22

2.4 Computational Order and Performance 22

2.4.1 Worst-Case Scaling 24

2.4.2 Experiments Demonstrating Algorithm Scaling 26

2.4.3 Comparison with Computer Algebra Systems 30

2.5 Summary 30

2.6 Supplementary Material 31

3 Enumeration Algorithm **33**

3.1 Introduction 33

3.2 Supercell selection and the Symmetry Group 35

3.3 Tree Search 37

3.3.1 Partial Colorings 37

3.3.2 The Stabilizers 41

3.3.3 Extension to Include Additional Degrees of Freedom 43

3.4 Conclusion 46

4 Conclusion **48**

4.1 Concluding Remarks 48

Bibliography **52**

List of Figures

2.1 Top row: All possible two-color colorings of a circle divided into 4 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. 8

2.2 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**). 14

2.3 (Color online) A recursive tree search for some of the possible matrices S for the problem of Section 2.2: 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 Eqn. (2.6). 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 that the recursive search is operating on. 20

2.4 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). 26

2.5 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. 27

2.6 Algorithm scaling as the number of elements in the finite set increases (for 2 colors). 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 algorithm's 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. 28

2.7 Normalized algorithm scaling with group size for an enumeration problem from solid state physics [7]. We used the unique permutation groups arising from all derivative superstructures of a simple cubic lattice for a given number of sites in the unit cell. The behavior is generally linear with increasing group size. 29

2.8 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 2.5. 30

3.1 (Color online) The empty lattice and 8 of the 36 configurations with only red atoms are shown for the example discussed in section 3.3. Above each partial coloring is a vector that indicates it's location in the tree, i.e. (x_r, x_y, x_p), where the x_is are integers that indicate which arrangement of that color is on the lattice and a $\bullet$ means that no atoms of that color have been placed. Below each configuration is either the label of a symmetrically equivalent configuration, along with the group operation that makes them equivalent, or the letters A and B. A and B are the branches that are built from the 1-partial colorings that are unique and are displayed in Fig. 3.2 . 35

3.2 (Color online) Here the A and B branches of the tree from Fig. 3.1 are shown. Each branch starts with the initial 1-partial coloring the branch is built from ($(0,\bullet,\bullet)$ and $(3,\bullet,\bullet)$ respectively). The branches then show a selection of the 2-partial colorings for that branch, and the unique full colorings that are found. As in Fig. 3.1 the vectors that indicate the configuration's location in the tree are displayed above the configurations. In the B branch configuration (3,19,0) is outlined for reference because it is used as an example later in the text. 38

3.3 (Color online) The configuration $(3,0,\bullet)$, shown on the left, is acted on by a reflection about the diagonal resulting in configuration $(3,6,\bullet)$, shown on the right. Because the symmetry group operation is a stabilizer for the configuration $(3,\bullet,\bullet)$ the red atoms were not affected. A stabilizer is a group element that leaves the set invariant. The yellow atoms, however, were mapped to a different configuration. This means we can use just the stabilizers for the $(3,\bullet,\bullet)$ configuration to compare all the 2-partial colorings of the form $(3,x_y,\bullet)$, where $(0 \leq x_y \leq C_y - 1)$, because any other group operation would map us to a different branch of the tree. 42

3.4 (Color online) To include displacement directions to the algorithm we represent the atoms to be displaced by a unique color and then convert them back once a unique configuration is found. In this figure two displaced yellow atoms are represented by red atoms until the previous portion of the algorithm is complete, then they are replaced by arrows again for the arrow enumeration. 45

List of Tables

2.1 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. . . . 15

Chapter 1

Introduction

1.1 Background

The discovery and application of new materials has always influenced the development of human civilization in a variety of ways. For example, the discovery of bronze brought on the creation of stronger metal alloys which in turn changed the balance of power among nations and resulted in the creation and destruction of empires. In more recent times, the development of steel has changed how the modern world operates: revolutionizing transportation, allowing for the construction of high-rise buildings, and improving household products like blenders and scissors. The past impact of new materials motivates ongoing searches for new materials today.

The process of finding new materials has advanced a great deal since the Bronze Age in which metallurgists tried to discover new materials by randomly combining known materials. Such a process is time consuming and success is infrequent. In the early 1900's Edison [8] and Ciamician [9] developed a "high-throughput" method of experimentally finding new materials. Their process involved mixing hundreds to thousands of samples at a time, increasing the rate at which new materials can be found. Since that time, the scientific community has developed a number of new tools that employ computational methods to perform first principles calculations to find new materials. These methods have dramatically advanced the field of material science and deserve some discussion.

One particularly powerful item in the computational material science toolbox came in 1964–1965 when Hohenberg and Kohn, quickly followed by Kohn and Sham, published work that set the foundation for Density Functional Theory (DFT). DFT allows scientists to calculate, from first principles, the energy of N interacting atoms. In general this is very difficult because it requires solving the Schrödinger equation for a nonlinearly coupled system of electrons. Due to this coupling of the electrons there are no analytical solutions of the Schrödinger equation for such many-body systems. Numerical solutions also fail because each electron has 3 coordinates in the wave function and numerical solutions require a discrete space for each coordinate. In order to model the wave function numerically using only 10 discrete points for each coordinate, we would need 10^{3N} points. This is a problem because if we have 27 particles in our system then the number of points we need exceeds the number of particles in the universe and cannot be stored in memory. DFT provides a way for the Schrödinger equation to be approximated and solved using density functions instead of wavefunctions. These density functions reduce the number of discrete points needed in the above example from 10^{3N} to 10^{3} since we only need to know the charge density from the electrons at any given point in space. This method allows scientists to determine the ground state properties of a material on a computer much faster than can be done in a lab.

Another advancement came in 1984, when Sanchez developed a basis for the Cluster Expansion (CE) method [10]. CE is a generalized Ising model which includes more interaction types, called clusters, than the nearest neighbor binary interactions used in the Ising model. Using these clusters it is possible to map the configurational energies of any configuration of a system that lives on a lattice. The governing equation of CE is:

$$E(\sigma) = \sum_f J_f \Pi_f(\sigma) \tag{1.1}$$

where σ is a particular configuration of atoms on the lattice, $E(\sigma)$ is the configurational energy, J_f is the effective interaction for cluster f, i.e., the f^{th} combination of lattice sites, and the sum runs over all possible clusters. Finally, the $\Pi_f(\sigma)$ is the "spin product" averaged over the entire lattice:

$$\Pi_{k,n}(\sigma) = \frac{1}{k}\frac{1}{M}\frac{1}{D_{k,n}}\sum_{I=1}^{M}\sum^{D_{k,n}} \sigma_{I_1}\sigma_{I_2}...\sigma_{I_k} \tag{1.2}$$

the $\sum^{D_{k,n}}$ runs over all unique cluster orientations where k is the number of vertices (i.e., atomic sites) in the cluster, n is the size of the cluster (the size is often a length or area measurement), M is the number of atomic sites in the unit cell, $D_{k,n}$ is the degeneracy due to the symmetries in the lattice, and the σ_{I_k} terms contains the spin occupation for the k^{th} vertex. Within the CE formalism each atomic species in a system will have a different spin value of σ_I.

Using equations 1.1 and 1.2 it is possible to predict the energy of crystal structures by expanding them in the CE basis once the effective interactions, J_f, for the system have been determined. Traditionally the J_f have been found using the genetic algorithm, which mimics the natural selection process to determine the effective interactions, that is trained on a set of energies calculated using DFT. That fit can then be used to determine the energies of configurations outside the DFT training set. For example, if we desire to investigate silver-platinum alloys, we can pick a handful of configurations of the two elements on the lattice and generate energy data using DFT. Then we can use the genetic algorithm to fit the DFT data and determine the Js. Once we know the J's we can look for new materials outside the original training set. With CE, we no longer have to perform DFT calculations for every configuration of the system to determine the structure of stable materials. Instead, we can use DFT data from a finite set of the possible configurations to train the

CE, which is then searched for stable configurations.

With DFT, we can investigate new materials faster than we can in a lab. However DFT is still limited as it can only explore one configuration at a time. But the space of configurations is countably infinite. Examining each of these one at a time would be a never-ending task that would produce few stable materials. CE overcomes this obstacle by allowing us to make predictions for a region of configuration space based on known data for only a few configurations in the region, thus enabling us to explore the space of possible materials at a much faster rate. However, no matter how fast CE is it still is only capable of exploring finite regions of materials space at a time since it cannot handle a countably infinite number of configurations.

Recently, the community has been developing high-throughput computational methods for materials science in order to create databases of materials properties that researchers can then search to find materials with desired properties. High-throughput computational methods are those that automatically scan over composition and structure space. The idea is to set up a program that will, given enough time, perform calculations for a large set of possible combinations of elements to determine their stability, and other properties, automatically. The advantage is that, rather than trying to invent new materials, one would simply log onto a database and search for materials with the desired characteristics. Many researchers have taken steps toward building databases that contain high-throughput DFT calculations. These include Ceder et al. [11] who founded the Materials Project, Curtarolo et al. [12] who made AFLOWLIB (Automatic-FLOW), and Saal et al. [13] the founders of OQMD (Open Quantum Materials Database).

As of yet, no high-throughput CE project is in operation. One of the goals of the materials simulation group here at BYU is to develop a a high-throughput CE code. The following work will increase the efficiency of our high-throughput CE code. The algorithms presented here have also been extended to include displacement directions in the enumeration of configurations, so that models to study the thermal properties of materials can also use this code to produce training structures.

1.2 Methods of Generating Derivative Superstructures

Over the years there have been many codes [14–18] written that are capable of finding all the symmetrically unique ways of arranging atoms on a lattice. The most common approach taken by these codes to find the symmetrically unique arrangements is to cycle through every possible arrangement of atoms and make comparisons between them using the symmetries of the lattice. This approach has been considered necessary in order to ensure the discovery of all the unique arrangements. However, the brute force search method that these codes employ is a shared shortcoming due to the computational cost. In some cases, this shortcoming actually limits the types of systems that can be enumerated because their size or atomic diversity (k-nary systems where $k > 2$) cause a combinatoric explosion in the number of possible structures.

This work describes two new algorithms: first a number theoretic implementation of Pòlya's counting theorem, and second, an efficient new algorithm for enumerating structures. These algorithms allow us to find the unique arrangements of atoms on a fixed lattice with known concentrations while only exploring a subset of the combinatorically possible arrangements. The algorithm for Pòlya's counting theorem predicts the correct number of unique configurations. This allows us to know if the system's unique configurations will fit in a computer's memory before the actual search begins. Additionally it serves as a test of the enumeration algorithm's accuracy.

The enumeration algorithm finds the unique configurations in a novel new way that greatly reduces the computational cost. This is accomplished by breaking the system up so that the possible arrangements of each atomic species are considered independently. This process resembles a tree search in which each level of the tree represents a unique atomic species. Using this tree search approach, large sections of the tree can be eliminated by skipping any partially occupied configurations that are symmetrically equivalent to other partially occupied configurations.

The full details of the Pòlya and enumeration algorithms are provided in the following chapters. The algorithms are presented in the order that they are used. That is the Pòlya algorithm is presented first followed by the enumeration algorithm for finding the unique derivative superstructures. The text of sections detailing the algorithms represent two seperate papers. The paper on the Pólya algorithm has been accepted for publication while the paper on the enumeration algorithm is going to be submitted soon.

Chapter 2

Pòlya’s Enumeration Algorithm

The following text is a paper that has been accepted for publication authored by Conrad W. Rosenbrock, Wiley S. Morgan, Gus L. W. Hart, Stefano Curtarolo, and Rodney W. Forcade. The authors contributions were the following: Conrad W. Rosenbrock was the creator of the algorithm, developed it into a user friendly package, and has written most of the text of this section; Wiley S. Morgan performed initial research into the possible design of the algorithm, translated the original Python code into Fortran, and performed extensive tests to check the algorithm’s performance; Gus L. W. Hart oversaw the development and testing of the algorithm, clarified the ideas surrounding the algorithm, and developed illustrative examples; Stefano Curtarolo reviewed the written paper; Rodney W. Forcade introduced the theorem to the group, wrote the mathematical analysis of its performance and scaling, and generated the necessary groups needed to rigorously test the algorithm. Rosenbrock, C. W., Morgan, W. S., Hart, G. L. W., Curtarolo, S., Forcade, R. W. 2016, Journal of Experimental Algorithmics (in press) 17 pages.

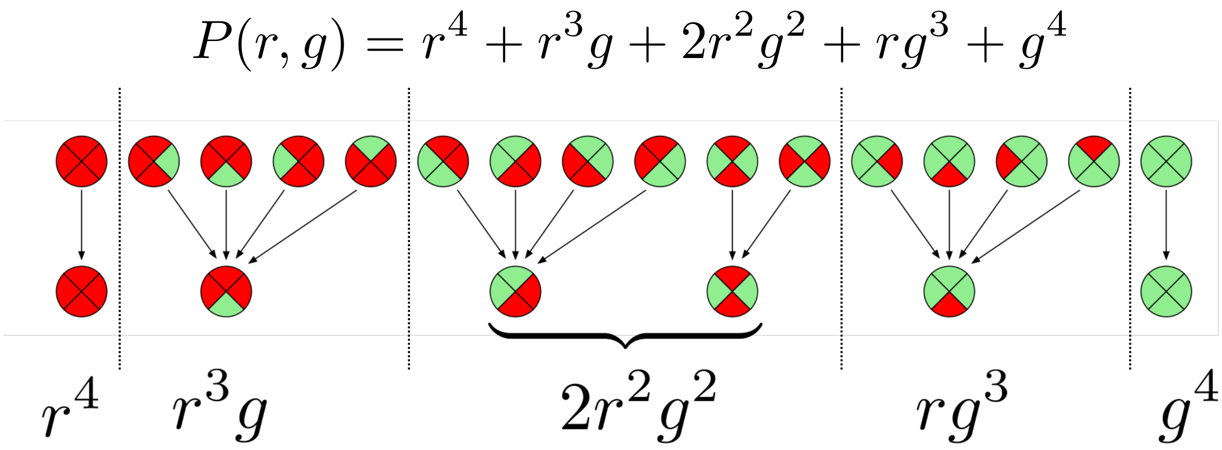

Figure 2.1 Top row: All possible two-color colorings of a circle divided into 4 equal sectors (left side of figure). Bottom row: All symmetrically-distinct binary colorings of the circle. Arrows indicate combinatorically-distinct colorings that are equivalent by symmetry.

2.1 Introduction

A circle partitioned into 4 equal sectors can be colored 16 different ways using two colors, $2^4 = 16$, as shown in Fig. 2.1. 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); or 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 [4, 5] 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, 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 Fig. 2.1. The Pólya theorem has shown its wide range of applications in a variety of contexts. Classically, it was applied to counting chemical isomers [5, 19, 20] and graphs [21]. Recent examples include confirming enumerations of molecules in bioinformatics and chemoinformatics [22, 23]; unlabeled, uniform hypergraphs in discrete mathematics [24]; analysis of tone rows in musical composition [25]; commutative binary models of Boolean functions in computer science [26]; generating functions for single-trace-operators in high energy physics [27]; investigating the role of nonlocality in quantum many-body systems [28]; and photosensitizers in photosynthesis research [29].

In computational materials science, chemistry, and related subfields such as computational drug discovery, combinatorial searches are becoming increasingly important, especially in high-throughput studies [30]. As computational methods gain a larger market share in materials and drug discovery, algorithms such as the one presented in this paper 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 Refs. [7, 31, 32], and Pólya's theorem was recently used in a similar crystal enumeration algorithm [33] that has been incorporated into the CRYSTAL14 software package [34].

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

[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.

two reasons. First, CASs are too slow for large problems that arise in a research setting, and secondly, generating the entire Pólya polynomial (which can have billions or trillions of terms) is unnecessary when one is interested in only a single stoichiometry.

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 2.2, followed by the algorithm for calculating the polynomial coefficients in Section 2.3. In the final Section, we investigate the scaling and performance of the algorithm.

2.2 Pólya Enumeration Theorem

2.2.1 Introduction to 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 Fig. 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 Fig. 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:

$$\begin{aligned} P(r,g) &= \frac{1}{4}\big((r+g)(r+g)(r+g)(r+g)+(r^4+g^4)+(r^4+g^4)+(r^2+g^2)(r^2+g^2)\big) \\ &= r^4+r^3g+2r^2g^2+rg^3+g^4. \end{aligned} \tag{2.1}$$

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 fifty sites to color, and twenty 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 one term out 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

$$P(x_1,x_2,\ldots,x_\xi) = \frac{1}{|G|}\Big(\sum_{\pi\in G} P_\pi(x_1,x_2,\ldots,x_\xi)\Big) \tag{2.2}$$

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 Section 2.1).

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, etc., 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 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 which 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:

$$\sum_{j=1}^{\xi} s_{ij} = k_i \qquad \text{(row sum condition);} \tag{2.3}$$

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

$$\sum_{i=1}^{t} r_i s_{ij} = c_j \qquad \text{(column sum condition),} \tag{2.4}$$

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

$$F(S) = \prod_{i=1}^{t} \binom{k_i}{s_{i1}, s_{i2}, \ldots, s_{i\xi}}, \tag{2.5}$$

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 (2.3) and (2.4).

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 straightforward. First, we must find all matrices S which meet the row and sum conditions (2.3) and (2.4) above. For each successful matrix, 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 (2) 1-cycles, (3) 2-cycles and (1) 4-cycle (so the number of sites is 12), and we have three colors with frequencies (red:green:blue → 4:6:2) respectively. Then we are looking for 3×3 matrices S whose rows sum to $(231)^T$ and whose columns (when dotted with the cycle-lengths $(124)^T$) sum to 4, 6 and 2 respectively. There are exactly five such matrices (see Figure 2.3 and discussion in Section 2.3):

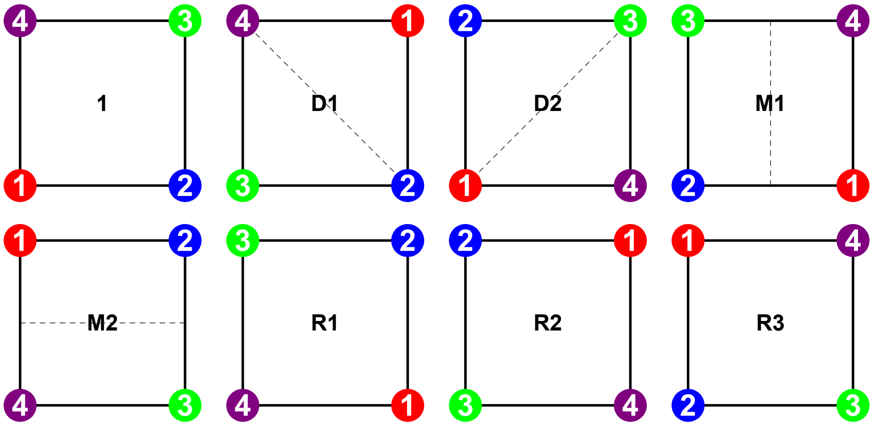

Figure 2.2 The symmetry group operations of the square. This group is known as the dihedral group of degree 4, or D$_4$. The dashed lines are guides to the eye for the horizontal, vertical and diagonal reflections (**M1**,**M2** and **D1**, **D2**).

$$\begin{pmatrix} 0 & 0 & 2 \\ 0 & 3 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 2 \\ 2 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 2 & 0 \\ 0 & 2 & 1 \\ 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 2 & 0 \\ 2 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 2 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} \qquad (2.6)$$

In each case the multinomial coefficient for the top row is $\binom{2}{2,0,0} = \binom{2}{2} = 1$ and for the bottom row is $\binom{1}{1,0,0} = \binom{1}{1} = 1$, 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.

2.2.2 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 2.2. These symmetries include four

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)	$x^4+ \quad +y^4$	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)	$x^4+ \quad +y^4$	0

Table 2.1 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.

rotations (by 0, 90, 180 and 270 degrees; labeled **1**, **R1**, **R2**, and **R3**) and four reflections (one horizontal, one vertical and two for the diagonals; labeled **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 2.1. 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 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 that it is part of. For a general r-cycle, the polynomial takes the form

$$(x_1^r + x_2^r + \cdots + x_\xi^r), \tag{2.7}$$

[2] The dihedral groups have multiple, equivalent names. The dihedral group of *degree* 4, D_4, is also called Dih_4 or the dihedral group of *order* 8 (D_8) within the various naming schemes.

for an enumeration with ξ colors. As described in section 2.2.1, 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 2.1. 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 $(6+2+2+2+2+0+2+0)/8 = 16/8 = 2$.

Next, we apply the procedure discussed in connection with equation (2.6) 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 one-cycles and one two-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 permutations requires a matrix $S = \begin{pmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{pmatrix}$ satisfying $s_{11} + s_{12} = 2$ and $s_{21} + s_{22} = 1$ (row sum condition (2.3)) and $s_{11} + 2s_{21} = 2$ and $s_{12} + 2s_{22} = 2$ (column sum condition (2.4)). There are only two matrices of non-negative integers satisfying those conditions simultaneously:

$$\begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \text{ and } \begin{pmatrix} 0 & 2 \\ 1 & 0 \end{pmatrix}. \tag{2.8}$$

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.

2.3 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 (2.3) and (2.4) to highlight the logical connections between our specific implementation and the general ideas from Section 2.2. 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

$$(x_1^r + x_2^r + \cdots + x_\xi^r)^d, \tag{2.9}$$

[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.

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

$$V = \{0, r, 2r, 3r, \dots, dr\}. \tag{2.10}$$

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

$$(x_1^r + x_2^r + \cdots + x_\xi^r)^d = \sum_{k_1, k_2, \dots, k_\xi} \mu_k \prod_{i=1}^{\xi} x_i^{rk_i} \tag{2.11}$$

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

$$k_1 + k_2 + \cdots + k_\xi = d. \tag{2.12}$$

As described in the introduction, the coefficients μ_k in the polynomial expansion Equation (2.11) are found using the multinomial coefficients

$$\begin{aligned}
\mu_k &= \binom{n}{k_1, k_2, \dots, k_\xi} = \frac{n!}{k_1! k_2! \cdots k_\xi!} \\
&= \binom{k_1}{k_1}\binom{k_1 + k_2}{k_2} \cdots \binom{k_1 + k_2 + \cdots + k_\xi}{k_\xi} \\
&= \prod_{i=1}^{\xi} \binom{\sum_{j=1}^{i} k_j}{k_i}.
\end{aligned} \tag{2.13}$$

Finally, we define the polynomial (2.7) for an arbitrary group operation $\pi \in \mathbf{G}$ as[4]

$$P_\pi(x_1, x_2, \dots, x_\xi) = \prod_{\alpha=1}^{m} M_\alpha^{r_\alpha}(x_1, x_2, \dots, x_\xi) \tag{2.14}$$

[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 of the form (2.9) 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

$$\sum_{\alpha=1}^{m} v_{i\alpha} = T_i, \tag{2.15}$$

where V_α is the set from Eq. (2.10) for multinomial $M_{\alpha}^{r_\alpha}$. At this point it is useful to refer to Figure 2.3 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 of cycles with length r_1. Constraint (2.15) is equivalent to the column sum requirement (2.4).

We first apply constraint (2.15) 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 (2.12). We can recursively examine each variable x_i in turn using these constraints to build a set of sequences

$$S_l = \{S_{l\alpha}\}_{\alpha=1}^{m} = \{(k_{1\alpha}, k_{2\alpha}, \ldots, k_{\xi\alpha})\}_{\alpha=1}^{m} \tag{2.16}$$

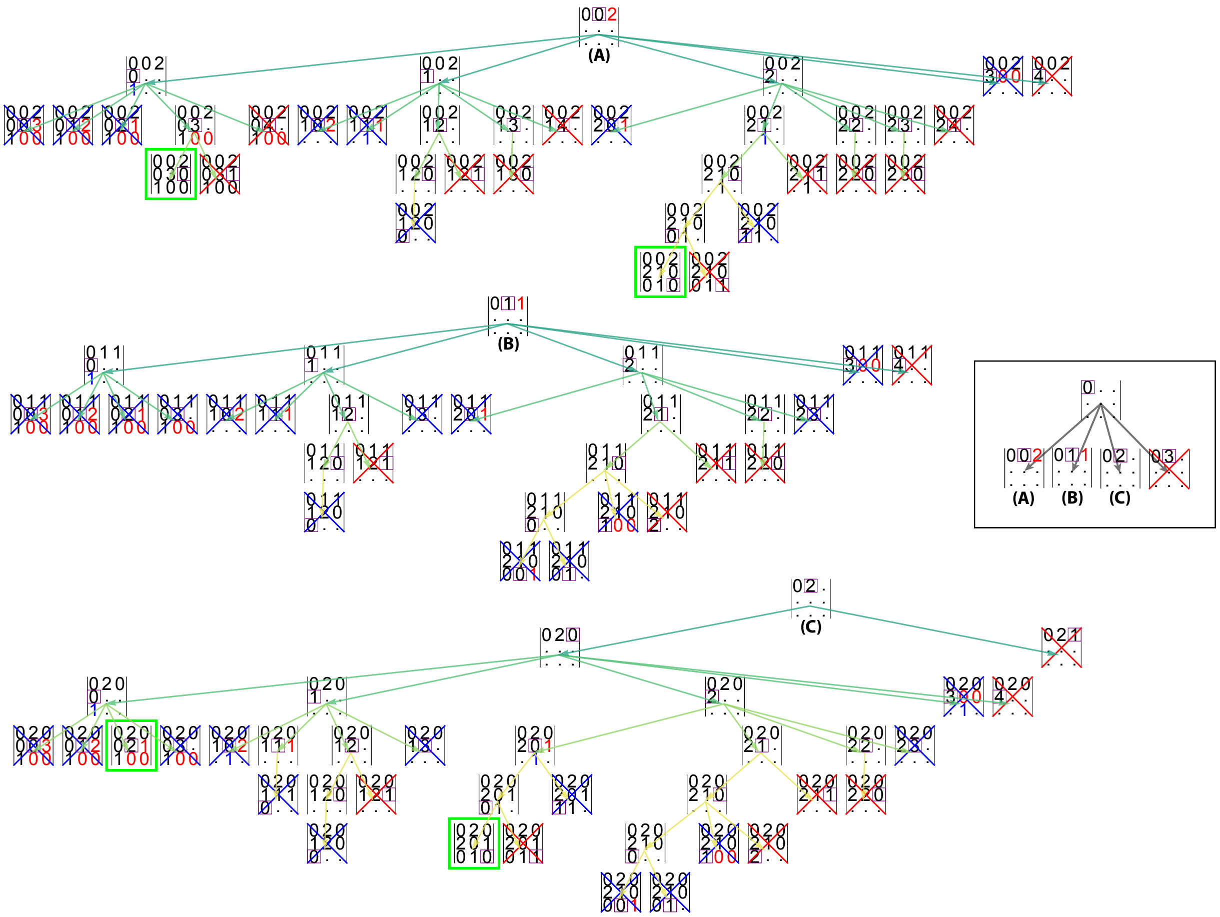

Figure 2.3 (Color online) A recursive tree search for some of the possible matrices S for the problem of Section 2.2: two 1-cycles, three 2-cycles and one 4-cycle. We have restricted the figure to include only the zero pendants of the tree, which produce four of the five relevant matrices in Eqn. (2.6). 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 that the recursive search is operating on.

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 maximum value of l depends on the target term T and how many possible $v_{i\alpha}$ values are filtered out using constraints (2.15) and (2.12) at each step in the recursion.

Once the set $\mathbf{S} = \{S_l\}$ has been constructed, we use Equation (2.13) 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

$$t_\pi = \sum_l \tau_l = \sum_l \prod_{\alpha=1}^{m} \binom{d_\alpha}{S_{l\alpha}}. \tag{2.17}$$

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 (2.15); 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), we end up with lots of possible ways to arrange them to sum to T_1 (which is not 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 (2.12) 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 compared to the target values T_1, T_2, etc. 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.

2.3.1 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. 3) using a Cartesian product over the sets in each multinomial.

When calculating multinomial coefficients, we use the form in Eq. (2.13) in terms of binomial coefficients with a fast, stable algorithm from Manolopoulos [35].

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.

2.4 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 2.3). 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

ALGORITHM 1: Recursive Sequence Constructor

```
Procedure initialize($i$, $k_{i\alpha}$, $M_\alpha^{r_\alpha}$, $V_\alpha$, $\mathbf{T}$)
    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 $x_i$ 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 $P_\pi$ (2.14).
    $V_\alpha$: possible exponents for $M_\alpha^{r_\alpha}$ (2.10).
    $\mathbf{T}$: $\{T_i\}_{i=1}^{\xi}$ target stoichiometry.
    ...........................................................................
    if $i = 1$ then
        $self$.used $\leftarrow$ $self$.root + $self$.parent.used
    else
        $self$.used $\leftarrow$ $self$.root
    end
    $self$.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 $\Leftarrow$ 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().
    ...........................................................................
    $sequences \leftarrow$ empty
    for $kid \in sequence.kids$ do
        for $seq \in$ expand($kid$) do
            $sequences \Leftarrow kid$.root $\cup\, seq$
        end
    end
    if $len(sequence.kids) = 0$ then
        $sequences \leftarrow \{kid$.root$\}$
    end
    return $sequences$
```

ALGORITHM 2: Recursive Sequence Constructor (Cont.)

```
Function build_$S_l$($\mathbf{k}, \mathbf{V}, P_\pi, \mathbf{T}$)
    Constructs $S_l$ from $\{k_{1\alpha}\}_{\alpha=1}^{m}$ for a $P_\pi$ (2.14).
    $\mathbf{k}$: $\{k_{1\alpha}\}_{\alpha=1}^{m}$ set of possible exponent values on the first variable in each $M_\alpha^{r_\alpha} \in P_\pi$.
    $\mathbf{V}$: $\{V_\alpha\}_{\alpha=1}^{m}$ possible exponents for each $M_\alpha^{r_\alpha}$ (2.10).
    $P_\pi$: Pólya polynomial representation for a single operation $\pi$ in the group $\mathbf{G}$ (2.14).
    $\mathbf{T}$: $\{T_i\}_{i=1}^{\xi}$ target stoichiometry.
    ..........................................................................................
    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)
    end
    return sequences
```

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. 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.

2.4.1 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 2.4). Numerical experiments confirm[5] that the

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

ALGORITHM 3: Coefficient Calculator

```
Function sum_sequences($S_l$)
    Finds $\tau_l$ (2.17) for $S_l = \{S_{l\alpha}\}_{\alpha=1}^{m}$ (2.16)
    $S_l$: a set of lists (of exponent sequences $\{k_{i\alpha}\}_{i=1}^{\xi}$) for each polynomial $M_\alpha^{r_\alpha}$ in $P_\pi$ (2.14).
    ..........................................................................................
    $K_l \leftarrow S_{l1} \times S_{l2} \times \cdots \times S_{lm} = \langle \{(k_{i\alpha})_{i=1}^{\xi}\}_{\alpha=1}^{m} \rangle_l$
    $coeff \leftarrow 0$
    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($\mathbf{T}$, $P_\pi$, $\mathbf{V}$)
    Constructs $\mathbf{S} = \{S_l\}$ and calculates $t_\pi$ (2.17)
    $\mathbf{T}$: $\{T_i\}_{i=1}^{\xi}$ target stoichiometry.
    $P_\pi$: Pólya polynomial representation for a single operation $\pi$ in the group $\mathbf{G}$ (2.14).
    $\mathbf{V}$: $\{V_\alpha\}_{\alpha=1}^{m}$ possible exponents for each $M_\alpha^{r_\alpha}$ (2.10).
    ..........................................................................................
    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
        $\mathbf{T} \leftarrow \text{sorted}(\mathbf{T})$
        $possible \leftarrow V_1 \times V_2 \times \cdots \times V_m$
        $coeffs \leftarrow 0$
        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_$S_l$($\{k_{1\alpha}\}_{\alpha=1}^{m}, \mathbf{V}, P_\pi, \mathbf{T}$)
                $coeffs \leftarrow coeffs +$ sum_sequences($S_l$)
            end
        end
        return $coeffs$
    end
```

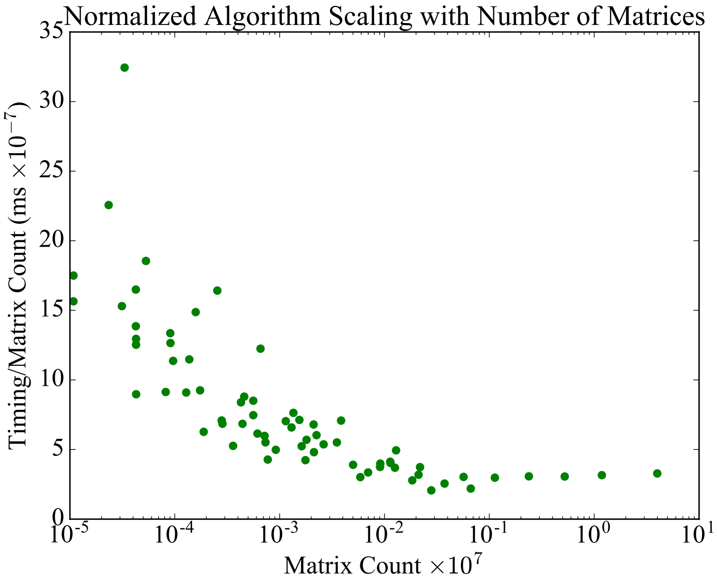

Figure 2.4 Normalized algorithm scaling with the number of relevant matrices to enumerate. For large matrix counts, the behavior appears linear, supporting the hypothesis that the algorithm scales roughly with the number of matrices. The scatter is appreciable only for small matrix counts (less than 10^6).

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 Eqn. (2.3)) 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 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, etc. This would be a very complex function to estimate, but we may expect it to grow exponentially for very large input. We did not find that to be an impediment for the sizes of problems we needed to solve.

2.4.2 Experiments Demonstrating Algorithm Scaling

In Figure 2.5, 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

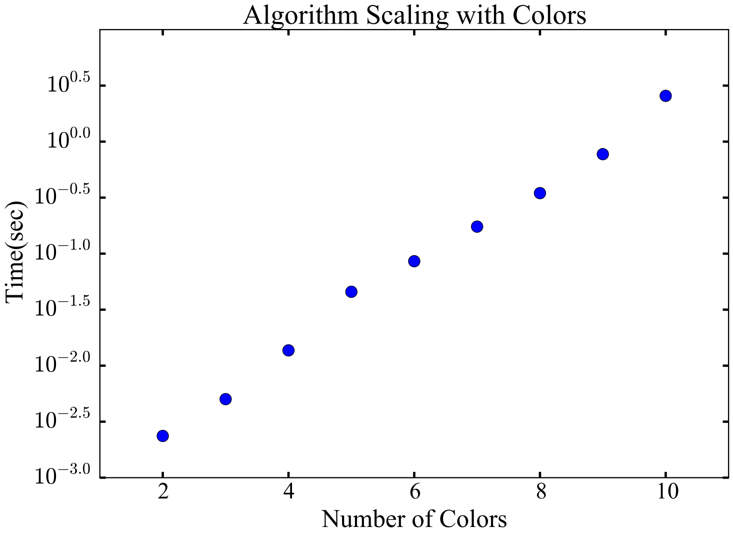

Figure 2.5 Log plot of the algorithm scaling as the number of colors increases. Since the number of variables x_i in each polynomial increases with the number of colors, the combinatoric complexity of the expanded polynomial increases drastically with each additional color; this leads to an exponential scaling. The linear fit to the logarithmic data has a slope of 0.403.

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 2.5 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 2.6, 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

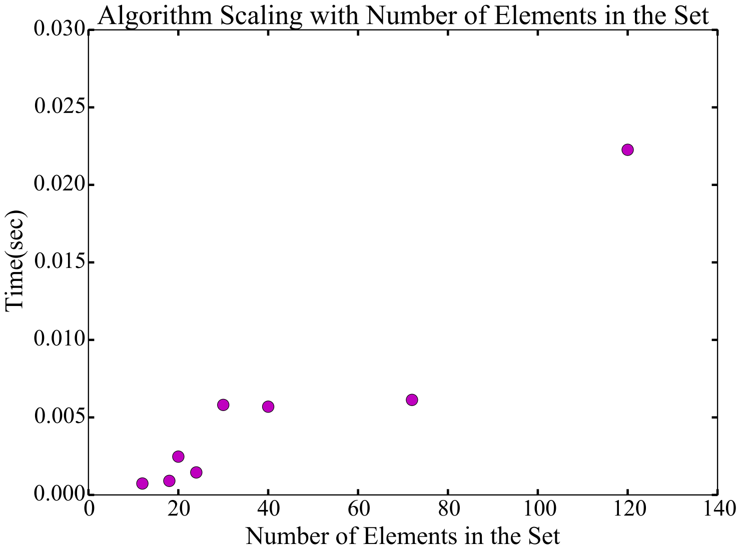

Figure 2.6 Algorithm scaling as the number of elements in the finite set increases (for 2 colors). The Pólya polynomial arises from the group operations' disjoint-cyclic form, so that more elements in the set results in a richer spectrum of possible polynomials multiplied together. Because of the algorithm's 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.

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. 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 [7]. 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 the specific disjoint-cyclic forms of the group operations. Even for groups with obvious similarity, the scaling may be different.

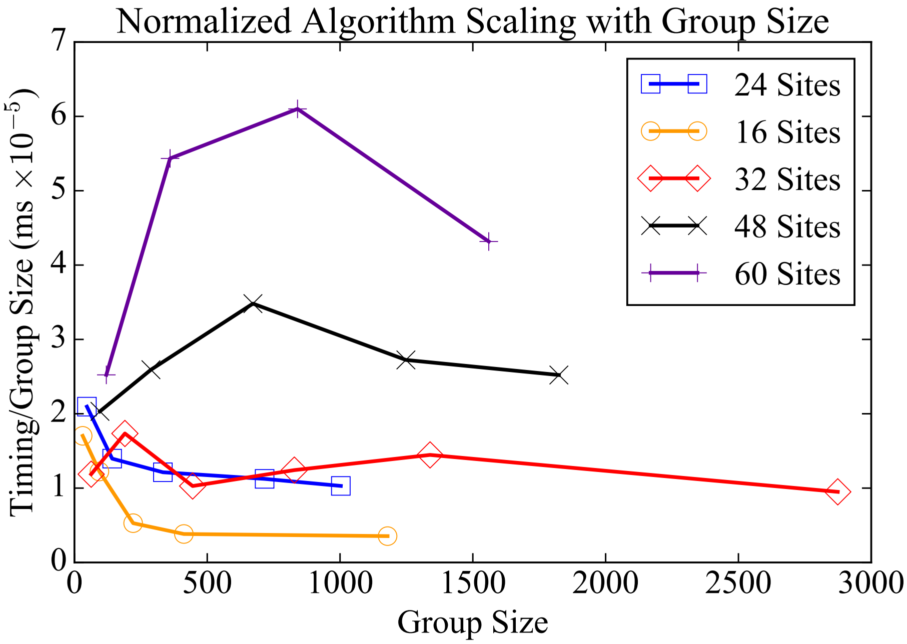

Figure 2.7 Normalized algorithm scaling with group size for an enumeration problem from solid state physics [7]. We used the unique permutation groups arising from all derivative superstructures of a simple cubic lattice for a given number of sites in the unit cell. The behavior is generally linear with increasing group size.

2.4.3 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 2.8.

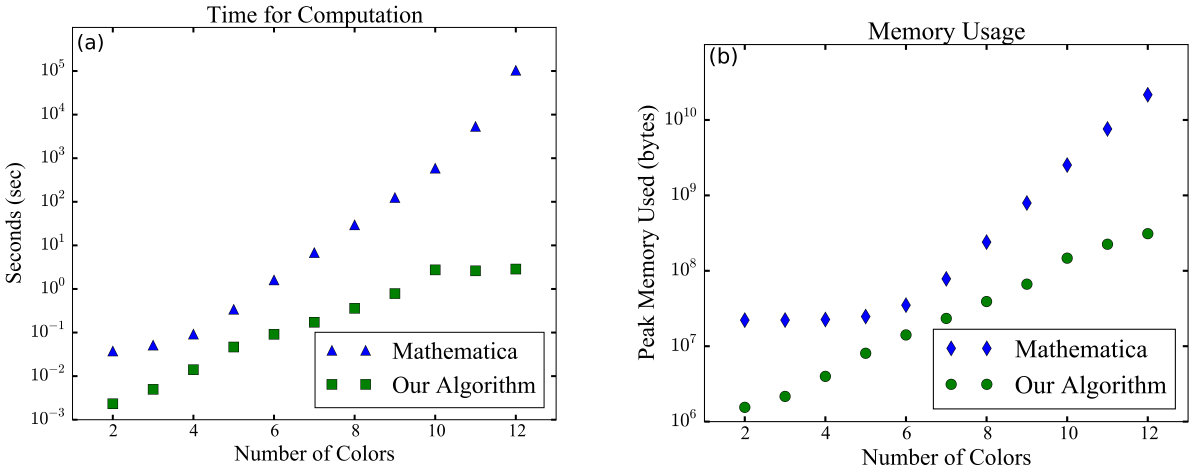

Figure 2.8 Comparison of the CPU time (a) and memory usage (b) between the Fortran implementation of our algorithm and Mathematica as the number of colors increases. These are the times needed to generate the data in Figure 2.5.

2.5 Summary

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

[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.

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.

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.

2.6 Supplementary Material

The source code implementing 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.

Chapter 3

Enumeration Algorithm

3.1 Introduction

In computational material science, one frequently needs to list the “derivative superstructures” [36] of a given lattice. A derivative superstructure is a structure with lattice vectors that are multiples of a “parent lattice” and have atomic basis vectors constructed from the lattice points of the parent lattice. For example, many phases in metal alloys are merely “superstructures” of fcc, bcc, or hcp lattices ($L1_0$, $L1_2$, B2, $D0_{19}$, etc.). When modeling alloys it is necessary to explore all possible configurations and concentrations of atoms within these superstructures. When determining if a material is thermodynamically stable, the energies of the unique arrangements are compared to determine which has the lowest energy.

Derivative superstructures are found using combinatoric searches [1–3, 14, 15, 17, 18], comparing every possible combination of atoms to determine which are unique. However, these searches can be computationally expensive for systems with high configurational freedom and are sometimes impractical due to the large number of possible arrangements.

The inefficiency of combinatoric searches makes finding the unique derivative superstructures a limiting factor in searches for high entropy alloys (HEA) [37–39]. The configurational complexity of HEAs prevents them from phase separating and this same complexity makes listing every possible arrangement of atoms impractical with current algorithms.

Other fields limited by the inefficacy of current enumeration methods include modeling materials that have disorder in their structures, such as site-disordered solids [40]. There are numerous techniques available for modeling these systems including cluster expansion (CE) [10] and a recently developed "small set of ordered structures" (SSOS) method. [41] However, the accuracy of these methods is still linked to the enumeration of the configurations being modeled. Increasing the number of configurations used to train the models can improve their predictive powers. Increasing the number of structures being used requires a more efficient enumeration technique than those currently available.

Leveraging the concepts of the algorithm presented in Ref. [3], we have developed an enumeration algorithm which has more favorable scaling in multinary cases than previous approaches. Our new algorithm uses a modified tree search to skip large numbers of symmetrically equivalent arrangements without generating any of the skipped configurations by using "partial colorings." Partial colorings are configurations in which the supercell is only partially occupied by the system's constituent atomic species. The partial colorings allow us to eliminate large classes of arrangements at once by determining if a partial coloring is unique and skipping the resultant branch of the tree if it is not.

Our algorithm also makes use of *stabilizers* from group theory. Stabilizers are a subgroup of the symmetry group that leave partial colorings invariant. Using the stabilizers, we reduce the number of group elements used to compare partial colorings further down in the tree.

The memory requirements of the search are greatly reduced by this algorithm's use of partial colorings and stabilizers. The increase in efficiency allows the algorithm to be extended to include discrete displacements in the enumeration. Enumerations with discrete displacements are used to

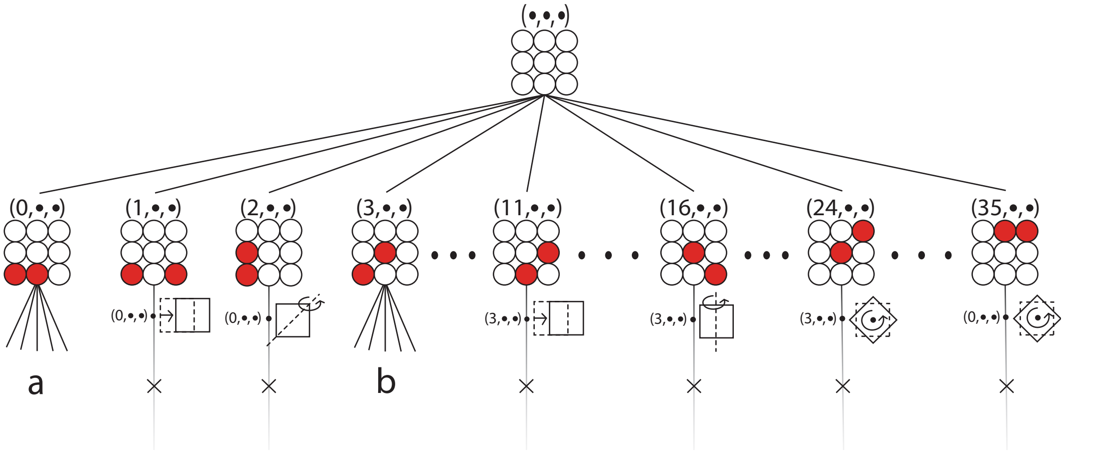

Figure 3.1 (Color online) The empty lattice and 8 of the 36 configurations with only red atoms are shown for the example discussed in section 3.3. Above each partial coloring is a vector that indicates it's location in the tree, i.e. (x_r, x_y, x_p), where the x_is are integers that indicate which arrangement of that color is on the lattice and a • means that no atoms of that color have been placed. Below each configuration is either the label of a symmetrically equivalent configuration, along with the group operation that makes them equivalent, or the letters A and B. A and B are the branches that are built from the 1-partial colorings that are unique and are displayed in Fig. 3.2

train phonon models [42, 43] that were previously too computationally expensive.

3.2 Supercell selection and the Symmetry Group

The first step in enumerating derivative superstructures is the enumeration of unique supercells. This step has already been solved by Hart and Forcade [1], but due to its importance to the algorithm a brief overview is provided.

The supercells, of size n, are found by constructing all Hermite Normal Form (HNF) matrices whose determinant is n. An HNF matrix is an integer matrix with the following form and relations:

$$\begin{pmatrix} a & 0 & 0 \\ b & c & 0 \\ d & e & f \end{pmatrix}, 0 \leq b < c, 0 \leq d < f, 0 \leq e < f, a, c, f > 0 \tag{3.1}$$

where $acf = n$. The HNFs determine all possible supercells for the system. For example, consider a 9-atom cell. In this case. $n = 9$ and a, c, f are limited to permutations of (1,3,3) and (1,1,9). Then, following the rules for the values of b, d, and e, every HNF for this system can be constructed. These HNFs represent all the possible supercells of size n of the selected lattice. Some of these are equivalent by symmetry, so the symmetry group of the parent lattice is used to eliminate any duplicates.

Next, we convert the symmetries of the lattice to a list of permutations of atomic sites. There is a one to one mapping between the symmetries of the lattice and atomic site permutations. The mapping from the symmetry operations to the permutation group is accomplished using the quotient group $G = L/L'$, where L is the lattice, constructed from the unit cell, and L' is the superlattice, constructed from the supercell. The quotient group G is found directly from the Smith Normal Form (SNF) matrices which can be constructed from the HNFs. This is done by solving the equation $UHV = S$ where H is the HNF, U and V are integer matrices with determinant ± 1 and S is the resultant SNF. The SNFs are diagonal matrices with positive integer entries where each diagonal divides the next one down. The group, G, is then $G = Z_{s_1} \bigoplus Z_{s_2} \bigoplus Z_{s_3}$, where s_i is i^{th} diagonal of the SNF and Z_{s_i} represents the cyclic group of order n.

Once the supercells have been found and their symmetry groups have been converted to the isomorphic permutation group, the algorithm can begin finding the unique arrangements of atoms within each supercell in a tree search framework. This is accomplished by treating each supercell with its symmetry group as separate enumeration problems. The results of the enumeration across all supercells are then combined to produce the full enumeration.

3.3 Tree Search

The enumeration algorithm resembles a tree search in which each branch corresponds to a specific configuration of atoms, many of which are not fully populated and are called partial colorings (see Fig. 3.2). The partial colorings are identified using a vector that indicates their locations within the tree. Once a partial coloring is constructed, the symmetry group operations that are stabilizers for that partial coloring are found. The stabilizers allow for the comparison of branches within the tree in a manner that minimizes the number of group operations used. These tools, partial coloring and stabilizers, are used to "prune" branches of the tree as they are being constructed, eliminating large classes of arrangements at once.

We will use a 2D lattice of 9 atoms as an illustrative example of the algorithm. The lattice will be populated with the following atomic species; 2 red atoms, 3 yellow atoms, and 4 purple atoms. A subset of the possible arrangements of this system are shown in Fig. 3.2. The concepts illustrated with this 2D example are equally applicable in 3D.

3.3.1 Partial Colorings

When searching for all unique configurations, it is useful to know a priori how many configurations are expected. Fortunately, a recently developed numerical algorithm for the Pólya enumeration theorem [4,5] allows us to do this quickly and cheaply. Use of the Pólya enumeration algorithm [6] determines the memory requirements of storing the unique arrangements. Also knowing how many configurations to expect confirms the algorithm's accuracy by verifying that the number found matches the prediction. For the 9 atom system considered here, the Pólya algorithm predicts that there are 24 unique arrangements to be found.

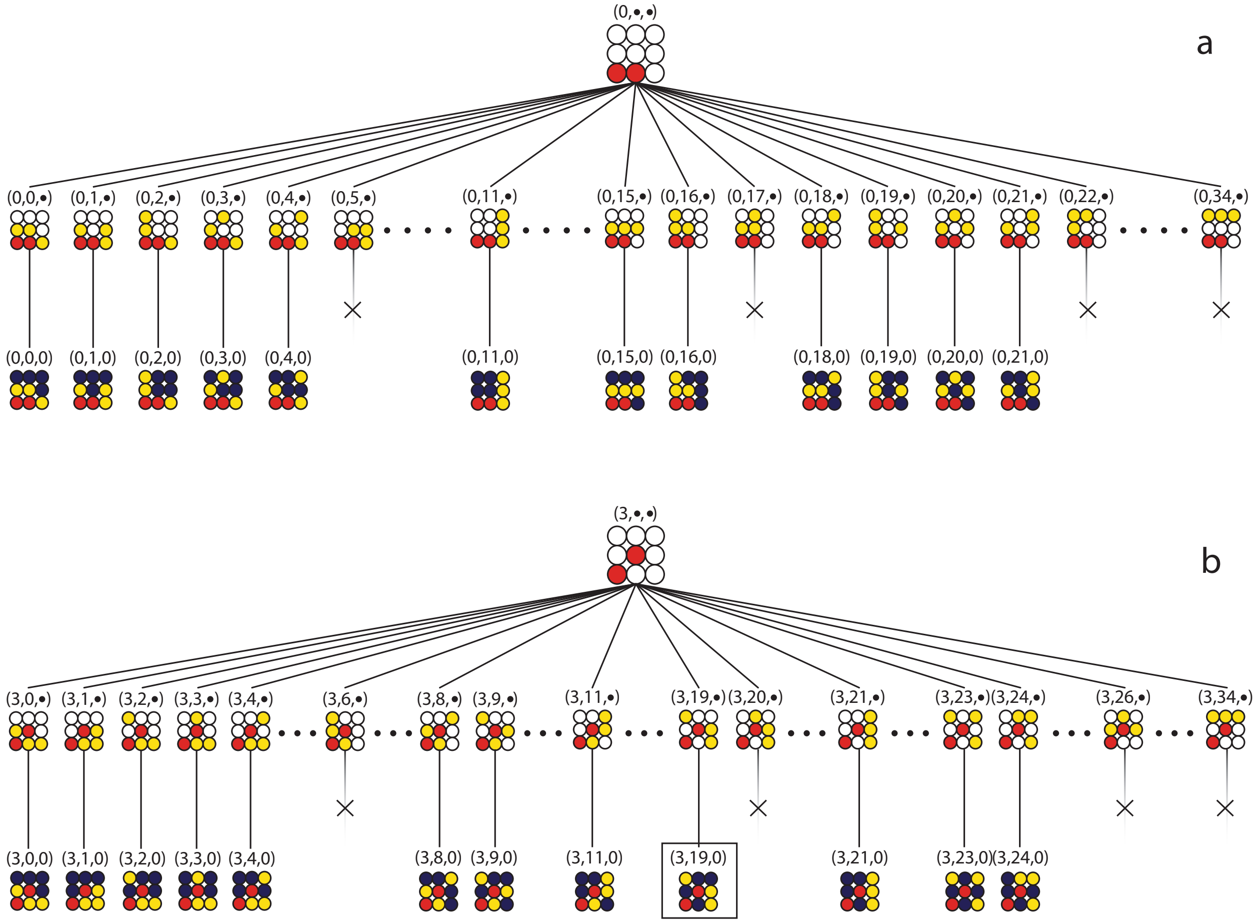

Figure 3.2 (Color online) Here the A and B branches of the tree from Fig. 3.1 are shown. Each branch starts with the initial 1-partial coloring the branch is built from ($(0,\bullet,\bullet)$ and $(3,\bullet,\bullet)$ respectively). The branches then show a selection of the 2-partial colorings for that branch, and the unique full colorings that are found. As in Fig. 3.1 the vectors that indicate the configuration's location in the tree are displayed above the configurations. In the B branch configuration (3,19,0) is outlined for reference because it is used as an example later in the text.

The algorithm places atomic species on the lattice according to their concentrations from lowest to highest. In this case the red atoms have the lowest concentration and are placed in the first two sites of the cell creating the first 1-partial coloring (a partial coloring is a configuration with only a subset of the atoms decorating the lattice). This is shown in the leftmost configuration, labeled $(0,\bullet,\bullet)$, in the second row of Fig. 3.1. The general procedure is to apply the symmetry group to each partial coloring in order to make comparisons between partial colorings and determine if they are symmetrically equivalent. For example in Fig. 3.1, the configuration labeled $(1,\bullet,\bullet)$ is equivalent to configuration $(0,\bullet,\bullet)$ by a translation of the lattice. At this stage we only have one partial coloring so it is unique and no comparisons need to be made, however the symmetry group is still applied to find the stabilizers described in section 3.3.2.

Comparisons between configurations are made by using a hash function. In computer science any data set can be placed in a hash table which associates a hash, or label, with the data. In our case, the configurations are listed within the hash table in the order they are created. The hash function then maps the configuration to a vector of integers with an entry for each species, color, in the system. The hash function used is similar to the one described in Ref. [3]. However, due to its importance in this algorithm, an overview of how the hash function works is provided.

The hash function for the algorithm uses the principles of combinatorics to uniquely identify each partial coloring using an integer vector. Its construction starts by finding the number of possible ways of arranging the colors on the lattice. This number can be found using the multinomial coefficient, which is equivalent to the product of binomial coefficients for each individual color:

$$\begin{gathered} C=\binom{n}{a_1,a_2,..,a_k}=C_1C_2...C_k= \\ \binom{n}{a_1}\binom{n-a_1}{a_2}...\binom{n-a_1-a_2-...-a_k}{a_k}, \end{gathered} \tag{3.2}$$

where n is the number of sites in the unit cell and $a_1,a_2,...,a_k$ are the number of atoms of species i

such that $\sum_i a_i = n$. The binomials determine the number of ways to place the atoms of each color within the lattice once the previous colors have been placed. By assigning each partial coloring an integer, x_i, from 0 to $C_i - 1$, where i is the color, we can build a vector that identifies the location, $(x_1, x_2, ..., x_k)$, of the configuration within the tree. For example, there are $C_r = \binom{9}{2} = 36$ ways to place the red atoms on the empty lattice. After the red atoms are placed there remain $C_y = \binom{7}{3} = 35$ ways to place the yellow atoms on the remaining lattice sites. This leaves $C_p = \binom{4}{4} = 1$ way to place the purple atoms on the lattice. Within Fig. 3.1 and 3.2 the vector locations have the form (x_r, x_y, x_p) and if the color has not been assigned yet then the x_is are replaced by dots indicating an empty vector site.

The hash function is a one to one mapping of the configurations to the location vectors. These numbers are constructed by considering each color separately and building a binary string of the color and the remaining empty lattice sites, where the color is a one and the empty site is a zero within the string. From the binary string we can then use a series of binomial coefficients to find the x_i's. The binomial coefficients are found by taking each 0 in the string that has 1's to the right of it and computing $\binom{p}{q-1}$, where p is the number of digits to the right and q is the number of 1's to the right of the 0. Summing this binomial for each zero that qualifies produces a number that tells us how many configurations came before the current one.

As an example of the hash function that constructs the location vector, consider configuration (3,19,0) of Fig. 3.2B. The construction begins with the red atoms represented as the following binary string (1,0,0,0,1,0,0,0,0) where every atom that is not red has been represented by a 0 and the red atoms by a 1. This string has 3 zeros that have a single 1 to their right, the first zero has 7 digits to its right, the second has 6 atoms to its right and the third has 5 atoms to its right. The resultant sum of binomials is $x_r = \binom{7}{0} + \binom{6}{0} + \binom{5}{0} = 1 + 1 + 1 = 3$. This result is the first entry in our location vector.

The second entry in the location vector is constructed for the yellow atoms. The bit string representation of the yellow atoms is (0,1,0,1,1,0,0), there are only 7 digits because the 2 red atoms have already been placed, so $x_y = \binom{6}{2} + \binom{4}{1} = 15 + 4 = 19$. The last entry in the location vector is built for the purple atoms which have the following bit string (1,1,1,1) so $x_p = 0$. The location vector is complete once all atoms within a configuration have been included.

The location vectors allow us to quickly determine if a configuration is unique by checking to see if the symmetry group maps the configuration to a configuration with a smaller location vector. This check is performed by applying the symmetry group one element at a time. The effect of the symmetry operations is to map the configuration's location to a second equivalent location. Uniqueness can then be determined by comparing the original and mapped locations for the configuration; if the mapped configuration has already be enumerated, that is if $x_{\text{original}} > x_{\text{mapped}}$, the configuration is not unique since it is equivalent to one we have already visited. For example, configuration $(2, \bullet, \bullet)$ shown in Fig. 3.1 can be turned into configuration $(0, \bullet, \bullet)$ by a 180 degree rotation about the diagonal. Since $(2, \bullet, \bullet)$ and $(0, \bullet, \bullet)$ are equivalent we conclude that $(2, \bullet, \bullet)$ is not unique because $2 > 0$. In summary, if any element of the symmetry group makes the vector "smaller", then the current configuration is not unique.

3.3.2 The Stabilizers

The use of the stabilizer subgroup increases the efficiency of the algorithm by reducing the number of symmetry operations needed to compare partial colorings. Each partial coloring has its own stabilizer subgroup. Fortunately, the stabilizers can be found when the symmetry group is applied to the partial colorings in the previous section, so finding the stabilizers costs nothing computationally. As an example of a stabilizer consider the cell $(3, \bullet, \bullet)$, displayed in Fig. 3.1, and reflect it about the diagonal. We can see that the red atoms are unaffected. This means that a reflection about the diagonal is a stabilizer for the 1-partial coloring $(3, \bullet, \bullet)$. In general, only a small subset of the symmetry group are stabilizers for any partial coloring.

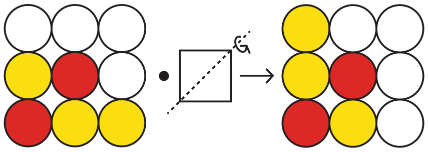

Figure 3.3 (Color online) The configuration $(3,0,\bullet)$, shown on the left, is acted on by a reflection about the diagonal resulting in configuration $(3,6,\bullet)$, shown on the right. Because the symmetry group operation is a stabilizer for the configuration $(3,\bullet,\bullet)$ the red atoms were not affected. A stabilizer is a group element that leaves the set invariant. The yellow atoms, however, were mapped to a different configuration. This means we can use just the stabilizers for the $(3,\bullet,\bullet)$ configuration to compare all the 2-partial colorings of the form $(3,x_y,\bullet)$, where $(0 \leq x_y \leq C_y - 1)$, because any other group operation would map us to a different branch of the tree.

The stabilizers leave the desired n-partial coloring the same, where n is the number of atomic species on the lattice. Once another color is added making an $(n+1)$-partial coloring the stabilizers for the n-partial coloring become the only group operations that can be applied without affecting the n-partial coloring, see Fig. 3.3. In other words, if we were to use any other group elements we would be comparing configurations that we already know are equivalent on the n-partial coloring level.

Once a unique n-partial coloring and its stabilizers have been found, the algorithm proceeds down the branch to the $(n+1)$-partial colorings, see Fig. 3.2. To check the uniqueness of the $(n+1)$-partial colorings the stabilizers from the n-partial coloring are used and any stabilizers for the $(n+1)$-partial colorings are stored. When a unique configuration is found on the $(n+1)$ level another color is added, making the $(n+2)$-partial colorings, and the process starts over again. The algorithm proceeds down a branch of the tree until a unique full configuration is found, such as (0,0,0) of Fig. 3.2. When the full configuration is found, the algorithm backs up one level and considers the next partial coloring. When no partial colorings are available on a level, the algorithm backs up until it finds a level with untested partial colorings. In this manner, the entire tree is explored while only sections with unique configurations are explored in detail.

For an example of the complete algorithm consider Figs. 3.1 and 3.2. The algorithm starts at $(\bullet,\bullet,\bullet)$ then builds the 1-partial coloring at $(0,\bullet,\bullet)$, which is unique by virtue of being the first partial coloring considered on this level, and records its stabilizers which are the only group operations that need to be applied on the 2-partial colorings (as seen in 3.3. The yellow atoms are then added to the configuration to build the 2-partial coloring at $(0,0,\bullet)$, of Fig. 3.2 A, which is also unique, and records its stabilizers. Next, it places the purple atoms to get the configuration at (0,0,0); this configuration is saved, then the algorithm backs up to the 2-partial coloring level to consider the configuration $(0,1,\bullet)$ and find its stabilizers. Once this process has been repeated for all 34 partial colorings in the vector $(0,x_y,\bullet)$ $(0 \leq x_y \leq 34 = C_y)$, the algorithm retreats to the 1-partial coloring level shown in the second row of Fig. 3.1 and finds that $(1,\bullet,\bullet)$ and $(2,\bullet,\bullet)$ are equivalent to $(0,\bullet,\bullet)$. It then begins to build the $(3,\bullet,\bullet)$ branch, of Fig. 3.2 B, in the same manner as the $(0,\bullet,\bullet)$ branch.

Since there are only two unique 1-partial colorings for this system, once both branches that originate from these 1-partial colorings have been explored the algorithm is complete. In the end, 24 unique configurations are found (shown in Fig. 3.2A and 3.2B), in agreement with the prediction from the Pòlya enumeration algorithm.

3.3.3 Extension to Include Additional Degrees of Freedom

Having established the algorithm, we will now address its extension to include displacement directions. In order to construct a basis for the space of displaced atoms we need only consider displacements along the axis of the cartesian coordinate system. These enumerations are more difficult because including displacement directions in the enumeration introduces a new degree of freedom into the system. Displacement directions are simply a way of indicating the direction that an atom could be displaced off the lattice. The enumeration of structures that include displacement directions can be used to build databases [11] of possible structures with displacements included.

Our algorithm changes only slightly if displacement directions are included in the enumeration. First, the atoms that will be displaced are treated as a different atomic species so that each displaced atom's unique locations can be determined (see Fig. 3.4 for an example where yellow displaced atoms are replaced with the red atoms from the example system used above). Once the arrows have been replaced by atomic species, the algorithm proceeds as normal until a full configuration is found. The algorithm then restores the arrows and uses the stabilizers of the full configuration to check for equivalent arrow configurations.

In order to determine if the combined arrow and color configuration is unique, each group element has to be paired with a second permutation that affects the arrows. The effect on the arrows is represented as a permutation of the numbers 0 to $d-1$ where each number represents a different displacement direction up to the d directions being considered. For example, if we consider the system in Fig. 3.4 we have two atoms being displaced along one of the 6 cardinal directions, then any arrow could have values of between 0 and 5 where each integer has an associated direction; up=0, right=1, down=2, left=3, into the page=4, and out of the page=5. The initial arrow vector, shown in the figure, is (up,up) and is represented as (0,0).

The comparison of the rotated and unrotated arrows is also achieved by use of a hash function. This function gives each arrow configuration a unique label that corresponds to the order it is constructed with the algorithm. This hash function takes a vector of arrow directions $(a_0, a_1, a_2, ..., a_k)$, where a_i is an integer from 0 to $d-1$ indicating the direction of the ith arrow and $k+1$ is the number of arrows, and finds:

$$x_a = \sum_{i=0}^{k} a_i d^i \tag{3.3}$$

This gives each arrow arrangement a unique integer label that we can then compare between symmetry operations. As was the case for the configurations, if the effect of a symmetry operation results in a relationship of $x_{\text{old}} > x_{\text{new}}$, then the arrow configuration is not unique and can be

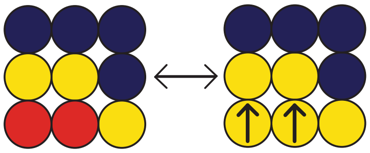

Figure 3.4 (Color online) To include displacement directions to the algorithm we represent the atoms to be displaced by a unique color and then convert them back once a unique configuration is found. In this figure two displaced yellow atoms are represented by red atoms until the previous portion of the algorithm is complete, then they are replaced by arrows again for the arrow enumeration.

ignored.

The stabilizers for the unique color configuration are used to map the arrows to new directions and the hash function is used to compare the original and mapped arrows. After an arrow arrangement is checked, the algorithm then increases the magnitude of the last a_k in the vector by 1 and checks it for uniqueness with the stabilizers. If increasing the magnitude of a_k would cause it to be greater than the value of $d-1$ then a_k becomes 0 again and a_{k-1} is increased by 1. This process is repeated until all the entries in the arrow vector are equal to $d-1$.

For example, the initial arrow vector for the system shown in Fig. 3.4 is (up,up) and is represented as (0,0). It is found to be unique since it is the first arrangement. For the next arrangement the arrow on the right is rotated to point to the right creating the arrangement represented as (0,1). This arrangement is also checked to see if it is unique. The rightmost arrow continues to be rotated every time a new arrangement is constructed until it is pointing out of the page and the arrangement represented as (0,5) has been considered. At this point all possible arrangements that have the first arrow pointing up have been considered so we reset the second arrow to point up and rotate the first arrow to make the arrangement (1,0). We then go back to increasing the last entry in the vector to create new arrangements in order to determine if any of them are unique until (1,5) is reached. The process is repeated until all possible arrangements, i.e., all 2-tuples of $0\ldots(d-1)$, have been considered. Once all the vectors have been considered, the algorithm goes back down the tree to find the next unique configuration of colors.

In this manner, discrete displacement directions can be added to the configurations without risk of memory overflow. In the example of this paper when the arrows are included in the enumeration the number of possible arrangements is 45360. However, the resultant number of unique arrangements, built by adding the arrows to the 24 configurations of Fig. 3.2, is only 663.[1]

3.4 Conclusion

Our previous algorithms [1–3] explored configuration space by comparing all possible configurations of the atoms to eliminate those that were symmetrically equivalent. The algorithm described here finds the same unique arrangements while only searching a subset of the combinatorically possible arrangements. With this enhancement we can now explore larger systems than our previous algorithm could handle. Additionally the new algorithm's efficiency allows for its extension to include displacement directions which can be used to train phonon models that predict the thermal properties of materials.

The algorithm's efficiency stems from its use of partial colorings of the lattice, in which only a subset of a system's atomic species populate the lattice, in a tree search algorithm. The partial colorings allow us to implement an indexing scheme that tracks where in the tree search we are. This then allows us to skip large sections of the tree that have symmetrically equivalent partial colorings. Skipping large classes of configurations in this manner avoids the combinatoric explosion of possible configurations which greatly reduces the memory requirements and runtimes for the enumeration.

The tree search also makes use of the stabilizer subgroup from group theory. The use of the stabilizer subgroup to compare partial colorings later in the tree greatly reduces the number

[1] This number can also be predicted by the Pòlya Enumeration Algorithm included within our code which has been modified slightly. The modifications allow for the algorithm to make predictions which include the extra degrees of freedom.

of symmetry operations used. Since using the stabilizers results in fewer comparisons being performed the overall runtime of the enumeration is also reduced. While this reduction in runtime is much smaller than the time saving from skipping over sections of the tree, its effects are still noticeable.

In order to include displacement directions in the enumeration we had to construct group actions that moved the atoms while changing the direction of displacement. To construct the group actions we combined each rotation with a second operation that changed the arrow directions so that each symmetry operation consists of two permutations; the first a permutation of the lattice sites, and the second a permutation of the displacement directions. These operations are then used within the algorithm to compare the possible displacement directions for the system on the fully populated lattice in order to determine which are unique.

With this new algorithm it is now possible to find the unique arrangements of systems for large systems, and systems with atomic diversity can be enumerated more efficiently. The code for the tree search portion of this algorithm is available for download [2].

[2] A python implementation of this code is available at https://github.com/wsmorgan/phonon-enumeration

Chapter 4

Conclusion

4.1 Concluding Remarks

A common problem in many fields involves enumerating the *symmetrically-distinct* colorings of a finite set, such as the derivative superstructures of a lattice in computational materials science. Within computational materials science the unique derivative superstructures are needed in order to determine the stable phases of a material. This is done by determining which superstructure has the lowest energy and will therefore be favored in nature. Other fields that benefit from knowing the symmetrically-distinct ways of arranging a set include the counting of chemical isomers [5, 19, 20] and graphs [21]; bioinformatics and chemoinformatics [22, 23]; unlabeled, uniform hypergraphs in discrete mathematics [24]; analysis of tone rows in musical composition [25]; commutative binary models of Boolean functions in computer science [26]; generating functions for single-trace-operators in high energy physics [27]; investigating the role of nonlocality in quantum many-body systems [28]; and photosensitizers in photosynthesis research [29].

In some of these applications it is not necessary to know what every possible unique arrangement of the set actually is, sometimes all that is needed is knowledge of the actual number of unique arrangements. In this context the Pólya counting theorem has been widely used in the past via Computer Algebra Systems (CAS) such as Mathematica. While these CAS approaches produce accurate results they can be slow for complex systems and are hard to incorporate into the increasing number of automated high-throughput codes. The low-level algorithm for the Pólya counting theorem presented here is much faster than the CAS algorithms and can easily be incorporated into the automated codes that are currently in use or under development making it extremely useful to for research groups in multiple fields.

In materials science (and related fields), however, a complete knowledge of the unique derivative superstructures is needed to determine materials properties. When all the unique arrangements of a set are needed it becomes necessary to use combinatoric searches to explore the space of possible arrangements. Combinatoric searches, which compare every possible combination of elements to determine which are unique, can be computationally expensive and are sometimes impractical. This is especially true when additional degrees of freedom, such as displacements from a lattice, are included in the enumeration.

The inefficiency of these combinatoric searches limits the accuracy of the predictive models like the Cluster Expansion (CE) because they limit the regions of configuration space included in the models. In order to improve the predictive power of models like CE, it is necessary to increase the region of configuration space we use to train them. The new enumeration algorithm presented here accomplishes that goal. This algorithm has more favorable scaling in multinary cases than previous approaches enabling researches to access larger regions of configuration space. This improvement was accomplished by using a modified tree search that skips large numbers of symmetrically equivalent arrangements without generating any of the skipped configurations. In this way this enumeration algorithm will allow researchers to improve the training models they use to make predictions about materials properties.

The increase in the algorithm's efficiency also allowed for its extension to include discrete displacements directions, the directions that an atom can be displaced from the lattice, in the enumeration. Enumerations with discrete displacements are used to train phonon models [42, 43]. With this extension of the algorithm, it is now possible to build databases [11] of structures with displacement directions that can be used by researchers to train the phonon models used to predict a material's thermal properties.

While both the Pólya algorithm and the enumeration algorithm have many potential applications independent of each other, together they represent a vast improvement in our ability to explore the configuration space of materials. This improvement will allow researchers to build better models for predicting the stable structures of materials. Additionally, the inclusion of displacement directions in the enumeration will allow researchers to build models for predicting the thermal properties of materials.

An example of how the paired algorithms will benefit materials science is that they give us more options for exploring the configuration space of materials, such as choosing to enumerate only a subset of the unique arrangements. For example, as a test of the new algorithm's performance we attempted to enumerate all the unique structures for a high-entropy alloy (HEA), HEAs are systems with 4 or more atomic species with equal concentrations [37–39]. The HEA chosen was a 20 atom system with 5 atomic species of equal concentration, for which there are 3×10^{11} possible configurations. A python implementation of the new algorithm was able to find all the unique arrangements for a single supercell of this system, around 10^9 unique arrangements, in 24 hours. Our previous algorithm was unable to run to completion due to the memory requirements. While this result is impressive, a list of 10^9 unique structures contains too much information to be useful for most applications. The Pólya algorithm paired with the new enumeration algorithm provides a way to produce a useful subset of the possible structures across multiple supercells and concentration ranges. This is accomplished by using the Pólya algorithm to determine the number of unique structures for each superstructure and concentration range desired and then enumerating a subset of the total number of structures following a carefully selected distribution. In this manner, systems with a large number of unique configurations that would normally be awkward or impossible to handle become much more tractable.

Implementing these algorithms into a CE code, such as UNCLE [44], will also bring us one step closer to a high-throughput CE code. Such a code would benefit the materials science community by increasing the rate at which materials could be explored and added to the existing databases, such as the Materials Project, for the use of the community at large. By including displacement directions in our enumeration model, this code also allows for the databases to include more information on these materials' thermal properties.

Bibliography

[1] G. L. W. Hart and R. W. Forcade, “Algorithm for generating derivative structures,” Computational Materials Science **77,** 224115 (2008).

[2] G. L. W. Hart and R. W. Forcade, “Generating derivative structures from multilattices: Algorithm and applications to hcp alloys,” Computational Materials Science **80,** 014120 (2009).

[3] G. L. W. Hart, L. J. Nelson, and R. W. Forcade, “"Generating derivative structures at a fixed concentration,” Computational Materials Science **59,** 101–107 (2012).

[4] G. Pólya and R. C. Read, Combinatorial Enumeration of Groups, Graphs, and Chemical Compounds (1987).

[5] G. Pólya, “Kombinatorische Anzahlbestimmungen fÃijr Gruppen, Graphen und chemische Verbindungen,” Acta Mathematica **68,** 145–254 (1937).

[6] C. W. Rosenbrock, W. S. Morgan, G. L. W. Hart, S. Curtarolo, and R. Forcade, “Numerical Algorithm for PÃşlya Enumeration Theorem,” (2015).

[7] G. L. W. Hart and R. W. Forcade, “Algorithm for generating derivative structures,” Phys. Rev. B **77,** 224115 (2008).

[8] G. Bryan, *Edison-The Man and His Work* (Bryan Press, 2007).

[9] G. Clamician, “The Photochemistry of the Future,” (1912).

[10] J. M. Sanchez, F. Ducastelle, and D. Gratias, “Generalized cluster description of multicomponent systems,” Physica A: Statistical Mechanics and its Applications **128,** 334–350 (1984).

[11] A. Jain *et al.*, “The Materials Project: A materials genome approach to accelerating materials innovation,” APL Materials **1,** 011002 (2013).

[12] S. Curtarolo, G. L. W. Hart, M. B. Nardelli, N. Mingo, S. Sanvito, and O. Levy, “The high-throughput highway to computational materials design,” Nature Materials **12,** 191–201 (2013).

[13] J. E. Saal, S. Kirklin, M. Aykol, B. Meredig, and C. Wolverton, “Materials Design and Discovery with High-Throughput Density Functional Theory: The Open Quantum Materials Database (OQMD),” JOM **65,** 1501–1509 (2013).

[14] A. Walle and G. Ceder, “Automating first-principles phase diagram calculations,” Journal of Phase Equilibria **23,** 348–359 (2002).

[15] A. van de Walle, M. Asta, and G. Ceder, “The alloy theoretic automated toolkit: A user guide,” Calphad **26,** 539–553 (2002).

[16] L. G. Ferreira, S. H. Wei, and A. Zunger, “Stability, Electronic Structure, and Phase Diagrams of Novel Inter- Semiconductor Compounds,” International Journal of High Performance Computing Applications **5,** 34–56 (1991).

[17] N. A. Zarkevich, T. L. Tan, and D. D. Johnson, “First-principles prediction of phase-segregating alloy phase diagrams and a rapid design estimate of their transition temperatures,” Physical Review B **75,** 104203 (2007).

[18] P. D'Arco, S. Mustapha, M. Ferrabone, Y. Noël, M. De La Pierre, and R. Dovesi, "Symmetry and random sampling of symmetry independent configurations for the simulation of disordered solids," Journal of Physics: Condensed Matter **25,** 355401 (2013).

[19] R. Robinson, F. Harry, and A. Balaban, "The numbers of chiral and achiral alkanes and monosubstituted alkanes," Tetrahedron **32,** 355–361 (1976).

[20] B. Kennedy, D. McQuarrie, and C. Brubaker Jr, "Group theory and isomerism," Inorganic Chemistry **3,** 265–268 (1964).

[21] F. Harary, "The number of linear, directed, rooted, and connected graphs," Transactions of the American Mathematical Society **78,** 445–463 (1955).

[22] K. Deng and J. Qian, "Enumerating stereo-isomers of tree-like polyinositols," Journal of Mathematical Chemistry **52,** 1581–1598 (2014).

[23] M. Ghorbani and M. Songhori, "The Enumeration of Chiral Isomers of Tetraammine Platinum (II)," Match-Communications in Mathematical and in Computer Chemistry **71,** 333–340 (2014).

[24] J. Qian, "Enumeration of unlabeled uniform hypergraphs," Discrete Mathematics **326,** 66–74 (2014).

[25] P. Lackner, H. Fripertinger, and G. Nierhaus, "Peter Lackner/Tropical Investigations," in *Patterns of Intuition* (Springer, 2015), pp. 279–313.

[26] A. Genitrini, B. Gittenberger, V. Kraus, and C. Mailler, "Associative and commutative tree representations for Boolean functions," Theoretical Computer Science **570,** 70–101 (2015).

[27] J. McGrane, S. Ramgoolam, and B. Wecht, "Chiral Ring Generating Functions & Branches of Moduli Space," arXiv preprint arXiv:1507.08488 (2015).

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

[29] M. Taniguchi, S. Henry, R. J. Cogdell, and J. S. Lindsey, "Statistical considerations on the formation of circular photosynthetic light-harvesting complexes from Rhodopseudomonas palustris," Photosynthesis Research **121,** 49–60 (2014).

[30] S. Curtarolo, G. L. W. Hart, M. B. Nardelli, N. Mingo, S. Sanvito, and O. Levy, "The high-throughput highway to computational materials design," Nature Materials **12,** 191–201 (2013).

[31] G. L. W. Hart and R. W. Forcade, "Generating derivative structures from multilattices: Application to hcp alloys," Phys. Rev. B **80,** 014120 (2009).

[32] G. L. W. Hart, L. J. Nelson, and R. W. Forcade, "Generating derivative structures for a fixed concentration," cms **59,** 101–107 (2012).

[33] S. Mustapha, P. D'Arco, M. D. L. Pierre, Y. Noël, M. Ferrabone, and R. Dovesi, "On the use of symmetry in configurational analysis for the simulation of disordered solids," Journal of Physics: Condensed Matter **25,** 105401 (2013).

[34] R. Dovesi *et al.*, "CRYSTAL14: A program for the ab initio investigation of crystalline solids," International Journal of Quantum Chemistry **114,** 1287–1317 (2014).

[35] Y. Manolopoulos, "Binomial Coefficient Computation: Recursion or Iteration?," ACM SIGCSE Bulletin InRoads 34 (2002).

[36] M. J. Buerger, "Derivative Crystal Structures," The Journal of Chemical Physics **15,** 1 (1947).

[37] J. W. Yeh, S. K. Chen, S. J. Lin, J. Y. Gan, T. S. Chin, T. T. Shun, C. H. Tsau, and S. Y. Chang, “Nanostructured High-Entropy Alloys with Multiple Principal Elements: Novel Alloy Design Concepts and Outcomes,” Advanced Engineering Materials **6,** 299–303 (2004).

[38] Y. Zhang, T. T. Zuo, Z. Tang, M. C. Gao, K. A. Dahmen, P. K. Liaw, and Z. P. Lu, “Microstructures and properties of high-entropy alloys,” Progress in Materials Science **61,** 1–93 (2014).

[39] M. C. Troparevsky, J. R. Morris, P. R. C. Kent, A. R. Lupini, and G. M. Stocks, “Criteria for Predicting the Formation of Single-Phase High-Entropy Alloys,” Physical Review X **5,** 011041 (2015).

[40] R. Grau-Crespo and S. Hamad, “The symmetry-adapted configurational ensemble approach to the computer simulation of site-disordered solids,” In *MOL2NET, International Conference on Multidisciplinary Sciences*, p. c002 (MDPI, Basel, Switzerland, 2015).

[41] C. Jiang and B. P. Uberuaga, “Efficient *Ab initio* Modeling of Random Multicomponent Alloys,” Physical Review Letters **116,** 105501 (2016).

[42] F. Zhou, W. Nielson, Y. Xia, and V. Ozolins, “Lattice Anharmonicity and Thermal Conductivity from Compressive Sensing of First-Principles Calculations,” Physical Review Letters **113,** 185501 (2014).

[43] J. W. Doak, C. Wolverton, and V. Ozolins, “Vibrational contributions to the phase stability of PbS-PbTe alloys,” Physical Review B **92,** 174306 (2015).

[44] D. Lerch, O. Wieckhorst, G. L. W. Hart, R. W. Forcade, and S. Müller, “UNCLE: a code for constructing cluster expansions for arbitrary lattices with minimal user-input,” Modelling and Simulation in Materials Science and Engineering **17,** 055003 (2009).