Computational Materials Science 136 (2017) 144–149

Contents lists available at ScienceDirect

Computational Materials Science

journal homepage: www.elsevier.com/locate/commatsci

Generating derivative superstructures for systems with high configurational freedom

Wiley S. Morgan [a,*], Gus L.W. Hart [a], Rodney W. Forcade [b]

[a] Department of Physics and Astronomy, Brigham Young University, Provo, UT 84602, USA
[b] Department of Mathematics, Brigham Young University, Provo, UT 84602, USA

ARTICLE INFO

Article history:
Received 5 January 2017
Received in revised form 11 April 2017
Accepted 16 April 2017

Keywords:
Enumeration
Derivative superstructures
Displacement patterns

ABSTRACT

Modeling 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 a combinatorial explosion when the supercell size is large, when there are more than two atom types, or when there are many displaced atoms. This problem persists even when there are only a relatively small number of unique arrangements that survive the elimination process. Here, we extend an existing algorithm to include the extra configurational degrees of freedom from the inclusion of displacement directions. The algorithm uses group theory and a tree-like data structure to eliminate large classes of configurations, avoiding the typical combinatoric explosion. With this approach we can now enumerate previously inaccessible cases, including atomic displacements.

1. Introduction

In computational material science, one frequently needs to list the “derivative superstructures” [1] 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 [2–8], 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 combinatoric explosion of possible arrangements.

Other problems impaired by the inefficiency of current enumeration methods include modeling materials that have disorder in their structures, such as site-disordered solids [9] or that include atomic displacements as a degree of freedom [10–12]. There are numerous techniques available for modeling these systems including cluster expansion (CE) [13] and a recently developed “small set of ordered structures” (SSOS) method [14]. However, the accuracy of these methods is still linked to the number of unique configurations being modeled. In other words, if the model is trained on a small set of configurations then it will not be able to make accurate predictions. Increasing the number of configurations used to train the models can improve their predictive power. Increasing the number of structures being used requires a more efficient enumeration technique than those currently available.

Leveraging the basic concepts of the algorithm presented in Ref. [6], we altered the algorithm to have more favorable scaling in multinary cases. The basic idea is to imagine the enumeration as a tree search and employ two new ideas: (1) “partial colorings” and (2) stabilizer subgroups. Section 3 illustrates the algorithm with a concrete example.

The concept of partial colorings is to skip entire branches of the tree that are symmetrically equivalent to previously visited branches. A partial coloring is an intermediate level in the tree (see Fig. 1) where configurations are not yet completely specified. It frequently happens that symmetric redundancy can be identified at an early, “partially colored” stage, avoiding the need to descend further down the tree.

* Corresponding author.
E-mail address: wiley.s.morgan@gmail.com (W.S. Morgan).

http://dx.doi.org/10.1016/j.commatsci.2017.04.015
0927-0256/© 2017 Elsevier B.V. All rights reserved.

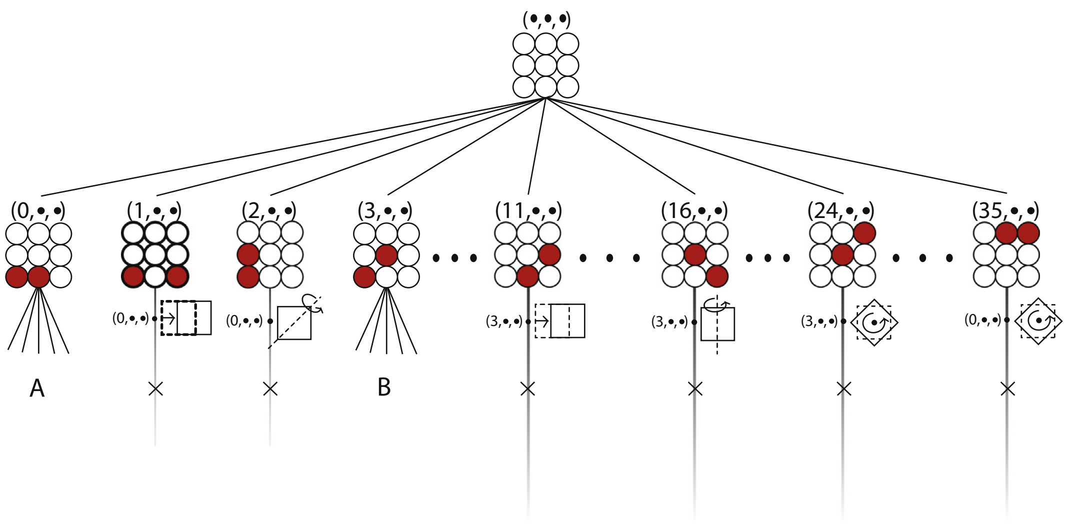

Fig. 1. The empty lattice and 8 of the 36 configurations with only red atoms are shown for the example discussed in Section 3. Above each partial coloring is a vector that indicates its location in the tree, i.e. (x_r, x_y, x_p), where the x_is are integers that indicate which arrangement of that color is on the lattice and a • means that no atoms of that color have been placed yet. 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. 2. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Stabilizer subgroups [15] further increase the efficiency of the new algorithm. The stabilizer subgroup at each stage is the set of symmetries which leave the current partial coloring unchanged. As we add more colors, and eliminate symmetrically equivalent colorings, we need not consider colorings which would be equivalent by non-stabilizer symmetries, since those colorings have already been implicitly eliminated. Note that the stabilizer subgroup will get smaller as we proceed down the search tree, thus simplifying and speeding our search.

2. Supercell selection and the symmetry group

The first step in enumerating derivative superstructures is the enumeration of unique supercells. This step was solved in Ref. [8], but due to its importance to the algorithm we provide a brief overview.

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:

$$\begin{pmatrix} a & 0 & 0 \\ b & c & 0 \\ d & e & f \end{pmatrix}, \; 0 \leqslant b < c, \; 0 \leqslant d < f, \; e < f \qquad (1)$$

where $acf = n$. The HNFs determine all possible the supercells for the system. For example, consider a 9-atom cell, then $n = 9$ 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, i.e., the groups are isomorphic. 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 via a standard algorithm using integer row and column operations. Thus $S = UHV$ where U and V are integer matrices with determinant ± 1 and S is the diagonal SNF matrix, where each positive integer diagonal entry divides the next one down. The group, G, is then $G = Z_{s_1} \oplus Z_{s_2} \oplus Z_{s_3}$, where s_i is ith 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 a separate enumeration problem. The results of the enumeration across all supercells are then combined to produce the full enumeration.

3. Tree search

Once a supercell has been selected, the remainder of the enumeration algorithm resembles a tree search. It is often possible to skip the descendents of a node because we know all its “leaves” will represent duplicate structures. These nodes represent incomplete configurations, or *partial colorings* (see Figs. 1 and 2). The partial colorings are identified using a “location vector” — a list of indices that specify the node in the tree. Once a partial coloring is constructed, the stabilizer subgroup for that partial coloring is found. The stabilizer subgroup allows for the comparison of branches within the tree in a manner that minimizes the number of group operations used. These tools (partial colorings and the stabilizer subgroup) are used to “prune” branches of the tree as they are being constructed, eliminating large classes of arrangements at once (Fig. 3).

We will use a 2D example of a 9-atom cell to illustrate 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 is shown in Fig. 2. The concepts illustrated with this 2D example are equally applicable in 3D.

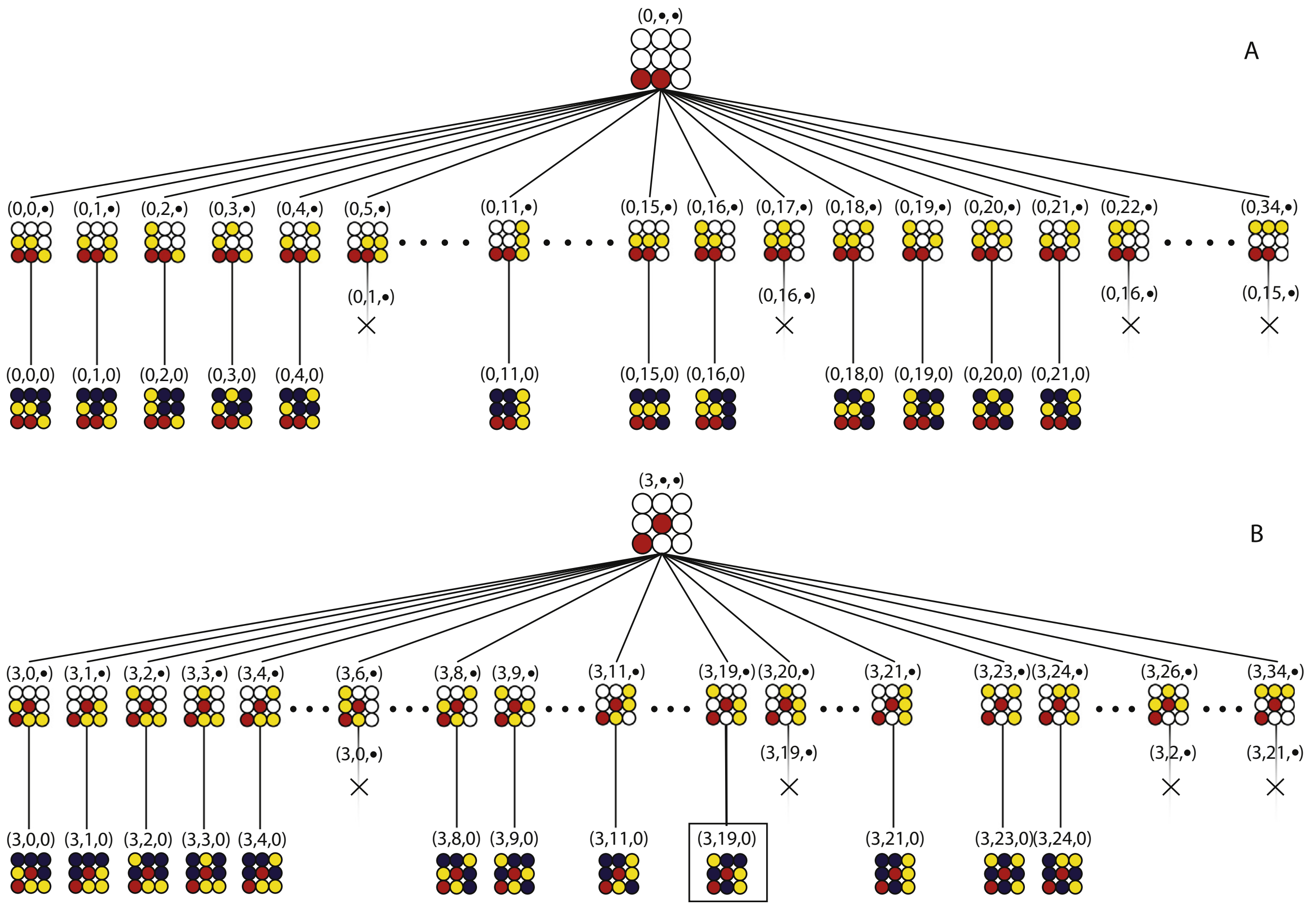

Fig. 2. Here the A and B branches of the tree from Fig. 1 are shown. Each branch starts with the initial 1-partial coloring the branch is built from ((0, •, •) and (3, •, •) respectively). The branches then show a selection of the 2-partial colorings for that branch, and the unique full colorings that are found. As in Fig. 1 the vectors that indicate the configurations location in the tree are displayed above the configurations and the symmetrically equivalent labels appears beneath them. In this figure the actions that make the configurations have been excluded due to their complexity. For example, The configuration labeled (0, 5, •) is equivalent to the (0, 1, •) configuration by a rotation about the vertical followed by a translation to the left. In the B branch configuration (3, 19, 0) is outlined for reference because it is used as an example later in the text. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

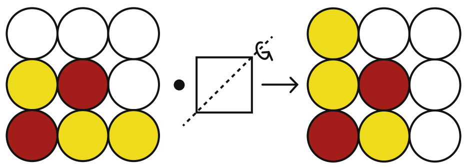

Fig. 3. The configuration (3, 0, •), shown on the left, is acted on by a reflection about the diagonal resulting in configuration (3, 6, •), shown on the right. Because the symmetry group operation is a stabilizer for the configuration (3, •, •) the red atoms were not affected. A stabilizer is a group element that leaves the set invariant. The yellow atoms, however, were mapped to a different configuration. This means we can use just the stabilizer subgroup for the (3, •, •) configuration to compare all the 2-partial colorings of the form $(3, x_y, \bullet)$, where $(0 \leqslant x_y \leqslant C_y - 1)$, because any other group operation would map us to a different branch of the tree. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

3.1. Partial colorings

When searching for all unique configurations, it is useful to know, a priori, how many configurations are expected. A recently developed numerical algorithm for the Pólya enumeration theorem [16–18] allows one to quickly determine the memory requirements of storing the unique arrangements. For the 9 atom system considered here, the Pólya algorithm finds 24 unique configurations (from the 1260 combinatorially distinct configurations).

The algorithm places atomic species on the lattice from smallest concentration to the largest. 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, •, •), in the second row of Fig. 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. 1, the configuration labeled (1, •, •) is equivalent to configuration (0, •, •) by a translation. At the (0, •, •) 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 stabilizer subgroup described in Section 3.2.

Comparisons between configurations are made by using a hash function. The hash function is a one-to-one mapping between the atomic configurations and the location vector. 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 uses the principles of combinatorics to uniquely identify each partial coloring using an integer vector. Its construction starts by determining the number of possible ways to arrange the colors on the lattice. The number of possible configurations can be found using the multinomial coefficient, which is equivalent to the product of binomial coefficients for each individual color:

$$C = \binom{n}{a_1, a_2, \dots, a_k} = C_1 C_2 \dots C_k = \binom{n}{a_1}\binom{n-a_1}{a_2} \dots \binom{n - a_1 - a_2 - \dots - a_k}{a_k}, \quad (2)$$

where n is the number of sites in the unit cell and $a_1, a_2, \dots, a_k$ 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, $(x_1, x_2, \dots, x_k)$, of the configuration within the tree. For example, there are $C_r = \binom{9}{2} = 36$ ways to place the red atoms (r) on the empty lattice. After the red atoms are placed then there remain $C_y = \binom{7}{3} = 35$ ways to place the yellow atoms (y) on the remaining lattice sites. This leaves $C_p = \binom{4}{4} = 1$ way to place the purple atoms (p) on the lattice. Within Figs. 1 and 2, the vector locations have the form (x_r, x_y, x_p) and if the color has not been assigned yet then the as-yet-unspecified x_is are indicated by dots.

The hash function maps a configuration to a location vector as follows: the x_i's 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 1 and the empty site is a 0 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 the binomials for qualifying zeros produces a number that tells us how many configurations came before the current one.

As an example, consider configuration (3,19,0) of Fig. 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 bit string (1,1,1,1), so $x_p = 0$. The location vector is complete once all colors have been included.

The location vectors allow us to determine if a configuration is unique by checking if an element of the symmetry group maps the configuration to another configuration with a *smaller* location vector. A symmetry operation maps a configuration's location to a second, equivalent location. Uniqueness is determined by comparing the original and mapped locations for the configuration; if the mapped configuration has already been enumerated, that is, if $x_{\text{original}} > x_{\text{mapped}}$, then the configuration is not unique because it is equivalent to one we have already visited. For example, configuration $(2, \bullet, \bullet)$ shown in Fig. 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 location vector "smaller", then the corresponding configuration has already been visited.

3.2. The stabilizer subgroup

The entire symmetry group does not need to be applied to a partial coloring; all that is needed is the stabilizer subgroup of the parent partial coloring (one level up the tree). The stabilizer subgroup is found when the symmetry group is applied to the partial coloring one level higher up the tree, so finding the stabilizer subgroup costs nothing computationally. As an example of an element of the stabilizer subgroup, consider the cell $(3, \bullet, \bullet)$, displayed in Fig. 1, and reflect it about the diagonal; the red atoms are unaffected. This means that a reflection about the diagonal is a member of the stabilizer subgroup for the 1-partial coloring $(3, \bullet, \bullet)$. In general, only a small subset of the symmetry group will be in the stabilizer subgroup for any partial coloring.

The stabilizer subgroup leaves the desired n-partial coloring unchanged, where n is the depth in the tree. When another color is added (making an $(n+1)$-partial coloring), the stabilizer subgroup for the n-partial coloring becomes the only group operations that can be applied without affecting the n-partial coloring. 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 stabilizer subgroup have been found, the algorithm proceeds down the branch to the $(n+1)$-partial colorings (see Fig. 2). To check the uniqueness of the $(n+1)$-partial colorings, the stabilizer subgroup from the n-partial colorings are used. (At this point, the stabilizer subgroup 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 is repeated until the final level of the tree is reached.

The algorithm proceeds down a branch of the tree until a unique full configuration is found, such as (0,0,0) of Fig. 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 but only sections with unique configurations are explored in detail.

For an example of the complete algorithm, consider Figs. 1 and 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 stabilizer subgroup. The yellow atoms are then added to the configuration to build the 2-partial coloring at $(0, 0, \bullet)$, of Fig. 2 A, which is also unique, and records its stabilizer subgroup. 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 stabilizer subgroup.

Once this process has been repeated for all 34 partial colorings in the vector $(0, x_y, \bullet)$ $(0 \leqslant x_y \leqslant 34 = C_y)$, the algorithm retreats to the 1-partial coloring level shown in the second row of Fig. 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 (Fig. 2 B) in the same manner as the $(0, \bullet, \bullet)$ branch. In this example, only 106 nodes of the 1296 are visited.

Since there are only two unique 1-partial colorings for this system the algorithm is complete once both branches that originate from these 1-partial colorings have been explored. In the end, 24 unique configurations are found (shown in Fig. 2A and

B), in agreement with the number determined by the Pòlya enumeration algorithm.

3.3. Extension to displacement degrees of freedom

Having established the algorithm, we will now address its extension to include displacement directions. These enumerations are more difficult because including displacement directions changes the action of the group. Displacement directions simply indicate 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 [19] 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. 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 stabilizer subgroup 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 set of permutations that determine how the symmetry operation 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. 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 achieved using a hash function different from the one used to hash the color configurations. This hash function takes a vector of arrow directions $(a_0, a_1, a_2, \ldots, a_k)$ and converts it to a unique integer label (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). The integer label is simply the mixed radix number:

$$x = \sum_{i=0}^{k} a_i d^i. \tag{3}$$

With the unique integer labels for each arrow arrangement, we can make comparisons 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 skipped.

The stabilizer subgroup 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 stabilizer subgroup. 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. 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 right most 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 the second arrow is set to point up and first arrow is rotated 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 the arrangements, i.e., all 2-tuples of $0, 1, \ldots, d-1$, have been considered. Once all the vectors have been considered, the algorithm goes back up the tree to find the next unique configuration of colors.

In this manner, discrete displacement directions can be added to the configurations. In this example, adding two arrows to the system increases the number of possible arrangements to 45,360 (the number of possible arrangements for the atoms alone is 1260). However, the resultant number of unique arrangements is only 663.

4. Algorithm scaling

Our new algorithm is much more efficient because it does not compare all possible configurations of atoms to determine which are unique. To demonstrate this improvement, we explored a typical use case for the algorithm by finding all unique configurations of atoms on an fcc lattice. We considered ternary and quaternary

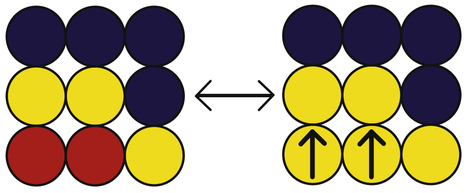

Fig. 4. To include displacement directions to the algorithm we represent the atoms to be displaced by a unique color and then convert them back once a unique configuration is found. In this figure two displaced yellow atoms are represented by red atoms until the previous portion of the algorithm is complete, then they are replaced by arrows again for the arrow enumeration. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

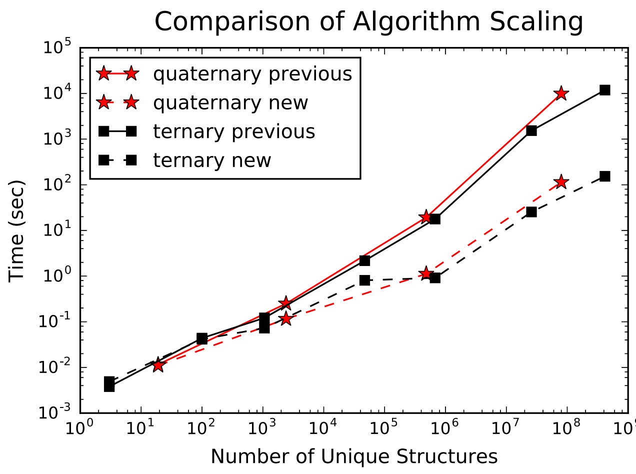

Fig. 5. The scaling of the new and previous algorithm for fcc ternary and quaternary systems in which the atomic species were of equal concentration. The ternary system was enumerated for cell sizes of 3–21 atoms and the quaternary system for cell sizes of 4–16 atoms.

systems in which all atoms were of equal concentrations. For the ternary system we explored cell sizes from 3 to 21 and for the quaternary system we considered cell sizes from 4 to 16. As can be seen in Fig. 5, the new algorithm is significantly faster (it scales better); for systems with approximately 20 atoms, the new algorithm is two orders of magnitude faster.

5. Conclusion

Our previous algorithms [6–8] explored configuration space by comparing all possible configurations of the atoms to eliminate those that were symmetrically equivalent. Our new algorithm uses a tree search and skips entire branches in the tree by employing partial colorings. Additionally, the stabilizer subgroups of the partial colorings increases the efficiency of comparing different configurations to determine which are unique. The new algorithm can be applied to cases where the previous algorithms failed because it handles cases with high configurational freedom more efficiently. This allowed us to extend it to include displacement directions.

With this new algorithm, it is now possible to find the unique arrangements of systems with higher configurational freedom. The systems now accessible include k-nary alloys, $k \geqslant 3$, and structures with displacement directions. This is accomplished by using an approach which closely resembles a tree search, but in which large classes of configurations are eliminated all at once. In this manner, we are able to partially avoid the combinatoric explosion which impedes the performance of the previous algorithms.

This algorithm has been implemented in the enumlib package and is available for public use at https://github.com/msg-byu/enumlib under the MIT License.

Acknowledgements

This research was funded by ONR grant MURI N00014-13-1-0635.

References

[1] M.J. Buerger, Derivative crystal structures, J. Chem. Phys. 15 (1) (1947) 1.
[2] P. D'Arco, S. Mustapha, M. Ferrabone, Y. Noël, M. De La Pierre, R. Dovesi, Symmetry and random sampling of symmetry independent configurations for the simulation of disordered solids, J. Phys. Condens. Matt. 25 (35) (2013) 355401.
[3] A. Van De Walle, G. Ceder, Automating first-principles phase diagram calculations, J. Phase Equilib. 23 (4) (2002) 348.
[4] A. Van De Walle, M. Asta, G. Ceder, The alloy theoretic automated toolkit: a user guide, Calphad 26 (4) (2002) 539–553.
[5] N.A. Zarkevich, T.L. Tan, D.D. Johnson, First-principles prediction of phase-segregating alloy phase diagrams and a rapid design estimate of their transition temperatures, Phys. Rev. B 75 (10) (2007) 104203.
[6] G.L.W. Hart, L.J. Nelson, R.W. Forcade, Generating derivative structures at a fixed concentration, Comput. Mater Sci. 59 (2012) 101–107.
[7] G.L.W. Hart, R.W. Forcade, Generating derivative structures from multilattices: algorithm and application to hcp alloys, Phys. Rev. B 80 (2009) 014120.
[8] G.L.W. Hart, R.W. Forcade, Algorithm for generating derivative structures, Phys. Rev. B 77 (2008) 224115.
[9] R. Grau-Crespo, S. Hamad, The symmetry-adapted configurational ensemble approach to the computer simulation of site-disordered solids, in: MOL2NET, International Conference on Multidisciplinary Sciences, page c002, Basel, Switzerland, December 2015. MDPI.
[10] S. Kadkhodaei, Q. Hong, A. van de Walle, Free energy calculation of mechanically unstable but dynamically stabilized bcc titanium, Phys. Rev. B 95 (2017) 064101.
[11] Z. Fei, W. Nielson, Y. Xia, V. Ozolins, Lattice anharmonicity and thermal conductivity from compressive sensing of first-principles calculations, Phys. Rev. Lett. 113 (18) (2014) 185501.
[12] K. Parlinski, Z.Q. Li, Y. Kawazoe, First-principles determination of the soft mode in cubic ZrO_2, Phys. Rev. Lett. 78 (1997) 4063–4066.
[13] J.M. Sanchez, F. Ducastelle, D. Gratias, Generalized cluster description of multicomponent systems, Phys. A 128 (1–2) (1984) 334–350.
[14] C. Jiang, B.P. Uberuaga, Efficient ab initio modeling of random multicomponent alloys, Phys. Rev. Lett. 116 (10) (2016) 105501.
[15] G.E. Edward, Transformation groups and C^*-algebras, Ann. Math. 81 (1) (1965) 38–55.
[16] G. Pólya, R.C. Read, Combinatorial enumeration of groups, graphs, and chemical compounds, Springer Science & Business Media, 2012.
[17] G. Pólya, Kombinatorische anzahlbestimmungen für gruppen, graphen und chemische verbindungen, Acta Math. 68 (1) (1937) 145–254.
[18] C.W. Rosenbrock, W.S. Morgan, G.L.W. Hart, S. Curtarolo, R.W. Forcade, Numerical algorithm for pólya enumeration theorem, J. Exp. Algorithm. 21 (1) (2016) 1–11.
[19] A. Jain, S.P. Ong, G. Hautier, W. Chen, W.D. Richards, S. Dacek, S. Cholia, D. Gunter, D. Skinner, G. Ceder, K.A. Persson, The Materials Project: a materials genome approach to accelerating materials innovation, APL Mater. 1 (1) (2013) 011002.