



Contents lists available at SciVerse ScienceDirect

Computational Materials Science

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

Generating derivative structures at a fixed concentration

Gus L.W. Hart^{a,*}, Lance J. Nelson^a, Rodney W. Forcade^b^a Department of Physics & Astronomy, Brigham Young University, Provo, UT 84602, USA^b Department of Mathematics, Brigham Young University, Provo, UT 84602, USA

ARTICLE INFO

Article history:

Received 17 November 2011

Received in revised form 7 February 2012

Accepted 9 February 2012

Keywords:

Alloys

Cluster expansion

Derivative structures

Enumeration

Ag–Pt

Silver–platinum

ABSTRACT

We present an algorithm for generating derivative superstructures for large unit cells at a fixed concentration. The algorithm is useful when partial crystallographic information of an ordered phase is known. This work builds on the previous work of Hart and Forcade [Phys. Rev. B **77** 224115, 2008; Phys. Rev. B **80** 014120, 2009]. This extension of the original algorithm provides a mapping from atomic configurations to consecutive integers when only a subset (fixed concentration) of all possible configurations is under consideration. As in the earlier algorithm, this mapping results in a minimal hash table and perfect hash function that enables an efficient method for enumerating the configurations of large unit cells; the run time scales *linearly* with the number of symmetrically-distinct configurations. We demonstrate the algorithm for a proposed structure in the Ag–Pt system comprising 32 atoms, with stoichiometry 15:17, a configuration space of $\sim 400,000$.

© 2012 Elsevier B.V. All rights reserved.

1. Introduction

The task of predicting the thermodynamically-stable structure of a material based on first-principles calculations has been an exciting prospect in computational materials science for many decades. The longevity of the problem stems from its potential impact, despite its difficulty. The problem is challenging because of the enormity of the search space—the sheer number of competing structures—and because of the relative high cost of computing first-principles energies. Algorithms for efficiently exploring crystal structure space and finding candidate minima are rich and varied. The general crystal structure prediction problem (CSP) has been discussed and reviewed in a number of other Refs. [1–10].

In this paper, we focus on a subset of CSP problems, the problem of predicting crystal structure when the underlying lattice is known. This narrower problem—the lattice decoration problem—is also an important one. Many of the ordered structures observed in intermetallic and semiconductor alloys are structures of this type. Codes for generating such *derivative structures* have been in existence for some time [11–15]. The list of derivative structures generated by these codes are frequently used to perform exhaustive searches for thermodynamically-stable phases.

Generating a complete list of *symmetrically-distinct* derivative superstructures, for an arbitrary parent lattice, can be done efficiently using permutation groups and an integer representation

of the superstructures [16,17]. The purpose of this paper is to extend the original algorithm of Refs. [16,17] to cases where, instead of enumerating all derivative superstructures, only those of a certain stoichiometry are generated.

There are practical computational materials science problems where the stoichiometry is naturally limited but the number of sites in the unit cell is large. In these cases, the original algorithm cannot be used because of the combinatorial explosion of configurations.

For example, one important class of lattice decoration problems is materials whose compositions are constrained by charge-neutrality considerations. When stoichiometric zirconia (ZrO_2) is doped with lower-valence cations such as Y^{3+} or Ca^{2+} , oxygen vacancies appear on the anion sublattice to maintain charge neutrality. Proper simulation of such systems requires unit cells with large numbers of atoms, which leads in turn to a prohibitively large number of configurations to consider. However, the concentration is fixed due to charge neutrality. This means that the actual number of relevant configurations is far less, but enumerating at a fixed concentration requires a new algorithm.

Complete enumeration is also impractical for many surface problems. For example, a binary alloy surface with one type of adsorbate leads to a quaternary enumeration problem where the number of configuration grows impossibly fast with increasing cell size. The old algorithm quickly becomes useless for these types of problems, but a fixed-concentration algorithm will be effective when coverages are small or alloy concentration ranges are narrow, as is often the case in experiment.

* Corresponding author. Tel.: +1 801 422 7444; fax: +1 801 422 0553.

E-mail address: gus.hart@gmail.com (G.L.W. Hart).

2. Algorithm recap

Although the details of the original algorithm are given in Refs. [16,17], we briefly review it so that the reader need not refer back to those references. In essence, the enumeration algorithm has two steps: (1) generating unique supercells and (2) generating unique labelings of those cells (labelings that are distinct under translational or rotational (orthonormal) symmetries of the parent lattice).

Step 1 is accomplished by representing each superlattice as an integer matrix in Hermite Normal Form (HNF), thus avoiding repetitions of the same lattice with different bases. We simply generate a list of all HNF's with a given determinant (corresponding to the size of the supercell). Depending on the rotational symmetries of the parent lattice however, some of the supercells defined by the list of HNFs may be equivalent to one another in a physical sense. These rotation duplicates can be removed by checking the equivalence of each pair of HNFs under each rotation of the parent lattice (see Fig. 1).

Step 2 begins by generating all possible labelings (atomic configurations) for each unit cell, as illustrated in Fig. 2. We do this by labeling the elements of the finite quotient group G (lattice over superlattice)—whose order is the same as the supercell but which we may think of as a list of elements on a line (as shown in Fig. 2). We then convert the symmetries of the lattice to a list of permutations of the group G , and use those permutations to eliminate labelings which are symmetrically equivalent, as illustrated in Fig. 3.

The example of Fig. 2 illustrates the concept. The list of binary configurations can be converted into a list of consecutive integers by substituting each label (green¹ or red) in each configuration with the numbers 0 and 1. The resulting binary numbers can be converted to base 10, forming a table of consecutive integers, as shown. Depending on the dimension and symmetries of the lattice, there may be different sets of permutations applied to that list, determining which labelings can be removed as redundant.

3. Extending the algorithm

The original algorithm generated all possible configurations for a given number of labels and label types, with repetitions allowed (four labels of two different types in the preceding examples). To generate, instead, all possible atomic configurations for a fixed concentration is equivalent to generating all permutations of a multiset. In the preceding examples, there were $n^k = 2^4 = 16$ possible configurations. If on the other hand, we were interested in all 4-atom permutations of two green and two red atoms (a fixed concentration), the multiset is $\{r, r, g, g\}$ and there are only $\binom{4}{2} = 6$ distinct configurations (i.e., permutations).

One might think that the solution to this problem is to use the same hash table and function from the previous algorithm and then discard unneeded configurations. This means that the hash table would be much larger than necessary, requiring an unworkable amount of memory. To enumerate all binary configurations in a unit cell of size fifty, for example, would require about $2^{50} \approx$ one million gigabytes, far exceeding the capacity of an average computer. In contrast, the hash table needed to enumerate all configurations in a 50 atom cell with concentration fixed at 20:80% requires only 10 gigabytes of memory. Our objective is a minimal hash table with consecutive integers and a mapping from each atomic configuration to its corresponding slot in the table. Such a hash table would enable us to do fixed concentration enumerations at much larger unit cell sizes than the original algorithm.

¹ For interpretation of color in Figs. 1–7, the reader is referred to the web version of this article.

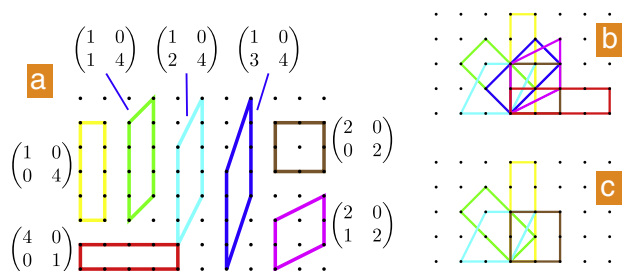


Fig. 1. (a) The 7 Hermite Normal Form (HNF) matrices of size 4, and their corresponding unit cells. (b) The same superlattices but depicted using the shortest (most orthogonal) basis vectors. (c) The four symmetrically-distinct superlattices of the seven. Under rotational symmetries of a square lattice (the parent lattice), the yellow 4×1 rectangle (vertical) is equivalent to the red one (horizontal), the blue rectangle is equivalent to the green, and the light blue is equivalent to the purple.

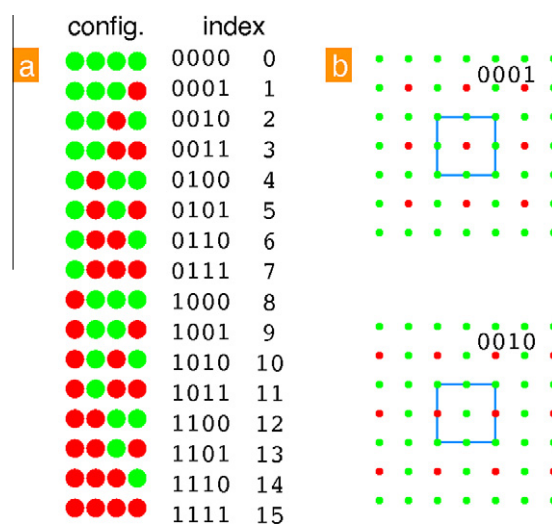


Fig. 2. (a) All possible binary combinations of 4 atoms. Because this is a binary case with 4 atoms, the total number of combinations is $k^n = 2^4 = 16$. By substituting the numbers 0 and 1, for the labels green and red, the combinations can be listed as sixteen 4-digit binary numbers which in turn can be converted to base 10. This mapping of configurations onto consecutive integers is a key element in implementing an efficient algorithm. (b) Two of the labelings, 0001 and 0010, applied to one of the 4-atom cells. By inspection we see that the two are crystallographically equivalent. The two cases are identical under a translation belonging to the parent lattice. Converting rotational and translational symmetries to simple permutations of the labelings makes it possible to quickly identify duplicates in the list of configurations.

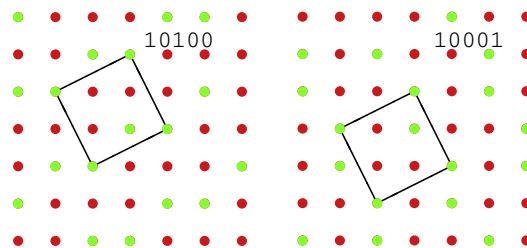


Fig. 3. An example of a labeling that is equivalent to the original under a rotational symmetry of the parent lattice. On the left-hand side, the original labeling of the unit cell is 10100 (or *grgr* going bottom to top, left to right). The superlattice (unit cell in black) is equivalent after a 90° rotation, but the labeling is permuted (right side, 10001, *grrg*), so the two labelings represent equivalent structures.

3.1. From configurations to integers

We will describe the algorithm in general terms but simultaneously work through an example, enumerating the “colorings”

of a superlattice with four lattice sites. We will represent the colorings (i.e., atomic configurations) of the three-dimensional unit cell using sequences of colors, as shown in Fig. 4. Note that this does *not* imply a one-dimensional lattice—the occupancy of sites in a supercell for a lattice of *any* dimension can be conveniently represented by a sequence of colors [18].

In the general case, we have k colors of items with a_1 items of color c_1 , a_2 of color c_2 , etc. The total number of items in our multiset is $n = a_1 + a_2 + \dots + a_k$. In the example, $n = 2 + 1 + 1 = 4$ ($a_1 = 2$ and $c_1 = \text{red}$; $a_2 = 1$ and $c_2 = \text{blue}$; and $a_3 = 1$ and $c_3 = \text{green}$).

The number of permutations of a multiset is the multinomial coefficient

$$C = \binom{n}{a_1, a_2, \dots, a_k} = \frac{n!}{a_1! a_2! \dots a_k!} = \binom{n}{a_1} \binom{n-a_1}{a_2} \dots \times \binom{n-a_1-a_2-\dots-a_{k-1}}{a_k} = \prod_{i=1}^k \binom{n-a_1-\dots-a_{i-1}}{a_i} \quad (1)$$

In other words, there are $C_1 = \binom{n}{a_1}$ ways to distribute the a_1 points of color c_1 , $C_2 = \binom{n-a_1}{a_2}$ ways to distribute the a_2 points of color c_2 among the remaining $n - a_1$ unoccupied points, and

$$C_i = \binom{n-a_1-a_2-\dots-a_{i-1}}{a_i} \quad (2)$$

ways to distribute the a_i points of color c_i , and $C = C_1 C_2 \dots C_k$. In the example of Fig. 4, we have

$$C = C_{\text{red}} C_{\text{blue}} C_{\text{green}} = \binom{4}{2} \binom{4-2}{1} \binom{4-2-1}{1} = 6 \times 2 \times 1 = 12.$$

Each color (i.e., label or atom type) can be distributed C_i ways. Let x_i represent which of those ways we choose to distribute color i , indexing from zero, so that $0 \leq x_i < C_i$. Then having indexed each color distribution in turn, we may index this particular permutation of our original multiset by the expression

$$y = f(x_1, x_2, \dots, x_k) = x_1 + C_1(x_2 + C_2(x_3 + C_3(\dots C_{k-2}x_{k-1}))). \quad (3)$$

Essentially we have created a mixed-radix number where the C_i are the multipliers and the x_i are the digits (i.e., place values). The function f is a mapping from the set of k -tuples (x_1, x_2, \dots, x_k) (with $0 \leq x_i < C_i$ for each i) to the set of integers y in the interval $\{0, 1, 2, \dots, C - 1\}$. The value y indexes our multiset permutations.

But how can we index the single-color distributions? That is, how do we produce the x_i 's in our function above? For each color,

we need to *order* the possible arrangements of that color among the remaining slots. We do this by noting that we may identify each such single-color arrangement as a binary string (0's and 1's). The length of each string is $m = a_i + a_{i+1} + \dots + a_k$ in which there are a_i ones and $m - a_i$ zeros. In Fig. 4, the first color is red and $m = 4$ and $a_1 = 2$ so there are $\binom{4}{2} = 6$ arrangements of red atoms and empty slots (and so $C_1 = 6$), as is apparent in the figure.

The standard algorithm for generating the single-color arrangements would be to successively look for the rightmost 1 with a zero to the right of it, move it one digit to the right, then pull all other 1's still to the right of it down against it. For example, 01001 would change (since the second one can be moved) to 00110. In the example, considering the red first, we have two ones and two zeros. We start with 1100 and proceed as follows.

1100
1010
1001
0110
0101
0011

We generate the x_i 's using a series of binomial coefficients. To compute the binomial coefficients, take each 0 which has 1's to the right of it, and associate the binomial coefficient $\binom{p}{q-1}$, where p is the number of digits to the right of it and q is the number of 1's to the right of it. Add all the coefficients, the result is x_i . What we're doing, in effect, is counting the number of configurations which *precede* the given one in our list—by counting how many would have the same digits to the left and a 1 (instead of a zero) in the current position.


To illustrate, here are two examples using Fig. 4. Consider the placement of the red atoms. The index for 0101 (numbers 4 and 10 in the figure) would be

$$x_{\text{red}} = \binom{3}{1} + \binom{1}{0} = 4$$

because we have one 0 with two 1's to the right (so $p = 3$ and $q - 1 = 1$) and one 0 with one 1 to the right (so $p = 1$ and $q - 1 = 0$). Similarly, the index for 0110 is

$$x_{\text{red}} = \binom{3}{1} = 3$$

because we have only one 0 with two 1's to the right.

As a complete example of how a configuration is mapped to an integer, take the configuration,  (numbered as 10 in Fig. 4). Recall that at the beginning of the section, we already determined $C_{\text{red}} = 6$, $C_{\text{blue}} = 2$; for green $C_{\text{green}} = 1$. For red, the binary string is 0101 so $x_{\text{red}} = \binom{3}{1} + \binom{1}{0} = 3 + 1 = 4$. For blue, there

are two slots left; the binary string is 01 so $x_{\text{blue}} = \binom{1}{0} = 1$. So,

$$y = x_{\text{red}} + C_{\text{red}} x_{\text{blue}} = 4 + 6(1) = 10. \quad (4)$$

Note from Eq. 3 that x_{green} is not needed. There is only one way to place the last color so it contributes nothing to the index y .

3.2. From integers to configurations

We also need the algorithm to work in reverse, to convert base-10 integers into configurations. The approach is shown diagrammatically in Fig. 5. One begins by first finding the x_i 's. This is done by taking the index of the configuration (y in Eq. 3) and dividing it successively by the C_i 's. Using the example of Fig. 4 again, take con-













0			0012	5
1			0102	11
2			0120	15
3	?		1002	29
4	new		1020	33
5	algorithm		1200	45
6	←		0021	7
7			0201	19
8			0210	21
9			2001	55
10			2010	57
11			2100	63

Fig. 4. The four-atom ternary configurations for a fixed concentration. Note that, because the concentration is fixed in this case, a simple extension (to base-3 numbers) of the hash table example given in the introduction does not result in consecutive integers. To utilize the group-theoretic approach for a fixed-concentration case, a new hashing scheme is required.

figuration number 9. Divide 9 by $C_1 = 6$. The quotient is 1 and the remainder is 3. So $x_{\text{red}} = 3$. Now take the quotient, 1, and divide by $C_2 = 2$. The quotient is 0 and the remainder is 1. So $x_{\text{blue}} = 1$. $x_3 = 0$ because there is always only one way to place the atoms of the last color (regardless of their multiplicity).

Now that we have the x_i 's, we need to convert each into the respective single-color configurations. As before, we will use a binary string representation, 1's for positions occupied by the current color and 0's for positions still empty. For the i -th color, there will be $m = n - a_1 - a_2 - \dots - a_{i-1}$ available positions and a_i of those positions will be occupied by 1's. Note that given the index number x_i for a string of length m with a_i ones, there will be a zero in the first (left) position iff $\binom{m-1}{a_i-1} \leq x_i$. Thus we may proceed, by a sort of *greedy algorithm* to determine whether each position contains a 1 or 0. In the latter case, subtract the binomial coefficient from the current index number and proceed. In other words:

Given x_i, m and a_i :

1. Let $I = x_i, t = a_i$ and $\ell = m$. (Now I, t and ℓ will be altered in the algorithm, but not x_i, a_i or m .)
2. If $\binom{\ell-1}{t-1} \leq I$ then let the $(m+1-\ell)$ th digit of our string be 0 and let $I = I - \binom{\ell-1}{t-1}$; otherwise, let the digit be 1 and set $t = t - 1$
3. Let $\ell = \ell - 1$. If $\ell > 0$, return to step 2; otherwise set all remaining digits to 0 and terminate.

As an illustration, consider the configuration number 9 in Fig. 4:

- $I = x_{\text{red}} = 3$ (as determined at the end of the previous section) and $t = a_{\text{red}} = 2, \ell = m = 4$.
- Now $\binom{\ell-1}{t-1} = \binom{3}{1} = 3$, which is $\leq I = 3$ so we set the first digit to 0 and set $I = I - 3 = 0$ and $\ell = \ell - 1 = 3, \ell > 0$ so return to step 2 again.

- $\binom{\ell-1}{t-1} = \binom{2}{1} = 2$, which is *not* $\leq I = 1$, so we set the second digit to 1 and set $t = t - 1 = 1$ and $\ell = \ell - 1 = 2, \ell > 0$ so return to step 2 again.
- $\binom{\ell-1}{t-1} = \binom{1}{0} = 1$, which is *not* $\leq I = 0$, so we set the third digit to 1 and set $t = t - 1 = 0$ and $\ell = \ell - 1 = 1, \ell > 0$, so return to step 2.
- For the last digit for the reds, $\binom{\ell-1}{t-1} = \binom{0}{-1} = 0$, which is $\leq I = 0$, so we set the last (fourth) digit to 0. $x_{\text{red}} = 3 \rightarrow 0110 \Rightarrow \text{○●●○}$.
- Next, find x_{blue} . There are two empty slots remaining and one blue atom. $I = x_{\text{blue}} = 1$ (from the end of the previous section) and $t = a_{\text{blue}} = 1, \ell = m = 2$.
- For the first digit of the blue configuration, $\binom{\ell-1}{t-1} = \binom{1}{0} = 1$, which is $\leq I = 1$ so we set the first digit to 0 and set $I = I - 1 = 0$ and $\ell = \ell - 1 = 1$.
- For the second (final) blue digit, $\binom{\ell-1}{t-1} = \binom{0}{0} = 1$, which is *not* ≤ 0 so the second digit should be a 1. That is, $x_{\text{blue}} = 1 \rightarrow 01 \Rightarrow \text{○●}$
- There is only one way (regardless of multiplicity) to place the final color. So green goes in the single remaining slot.
- $\text{○●●○} + \text{○●} + \text{○} \Rightarrow \text{○●●○●○}$

4. Application

Here we illustrate the application of the algorithm by a real-world example. In 1996, Durussel and Feschotte re-examined the Ag–Pt phase diagram in an effort to confirm previously-claimed phases that were somewhat surprising. Their rather extensive work resulted [19] in a significant redrawing of the phase diagram and, in contrast to previous work, they found that there was only *one* stable ordered-phase.

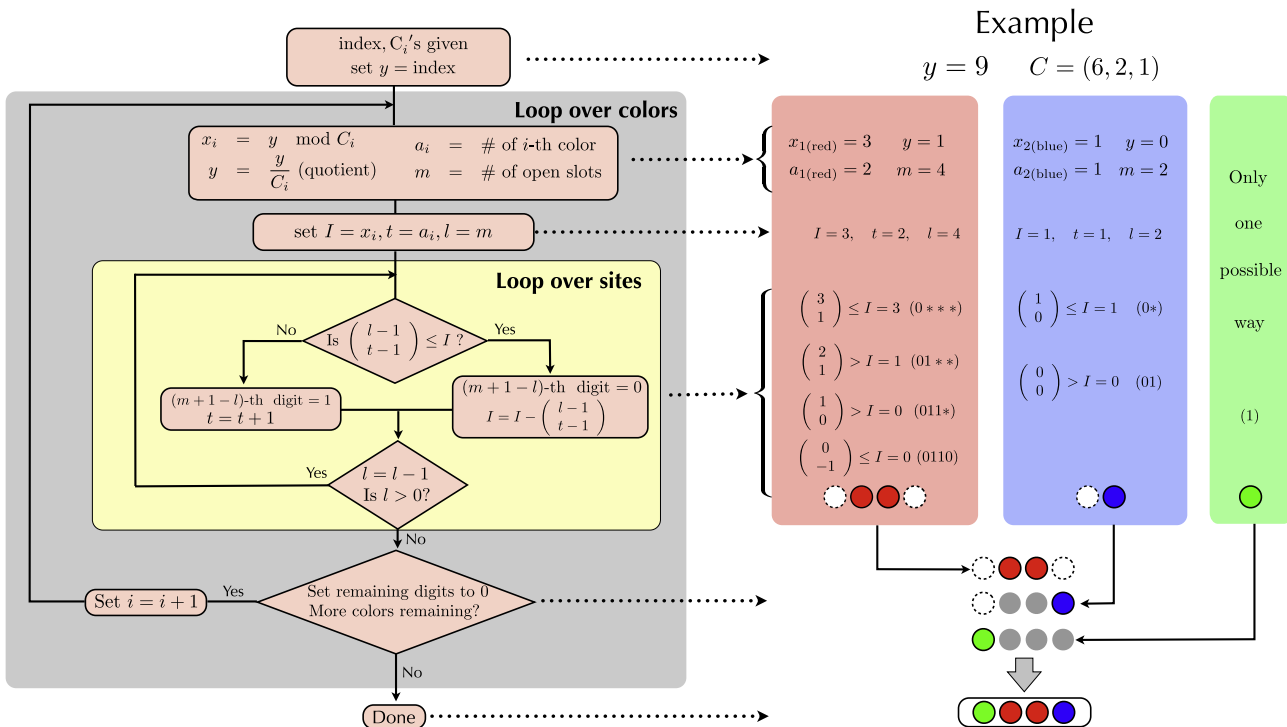


Fig. 5. Flow diagram of the algorithm discussed in Section 3.2.

That ordered phase was reported to have a stoichiometry of 53 at.% Pt, and X-ray analysis of a sample containing this new phase indicated only one lattice parameter value larger than the fcc lattice parameter. The lattice parameter value was ~ 8.0 Å, approximately twice the value of the fcc phases. Consequently, the authors conjectured that the unit cell of the new phase was a 32-atom supercell (a $2 \times 2 \times 2$ multiple of the conventional fcc cell) with the surprising stoichiometry of 15:17. They were unable to further characterize the structure of the new phase.

If one could rapidly compute the energy of a configuration (via a cluster expansion or other fast Hamiltonian), then one could take the list of all distinct 15:17 configurations inside the conjectured unit cell, calculate an energy for each one, and quickly determine which was most stable (and presumably the experimental structure). With the original enumeration algorithm, this is not practical because one must enumerate all symmetrically-distinct configurations (of all concentrations) in the cell. But for this case we only need the subset with stoichiometry 15:17. In the original algorithm, this requires us to reduce the long $2^{32} \approx 10^{10}$ list of configurations to the symmetrically-distinct set. With the new algorithm, we need only reduce a list of $\binom{32}{17} \approx 5 \times 10^8$, 20 times smaller.

In the former case, the algorithm is at the memory limit of a typical computer (~ 10 gigabytes). Additionally, the full enumeration would have required nearly 1 week of cpu time. Saving a factor of 20 removes the memory constraint and also speeds up the enumeration considerably. In the more typical cases discussed in the Summary, the differences between the full enumeration and concentration-restricted enumeration will be even larger.

The number of symmetries of the 32-atom unit cell is $48 \times 32 = 1536$ (translations and rotations) so the starting list of configurations is reduced by almost this factor, leaving only about 400,000 symmetrically-distinct configurations. Using a cluster expansion constructed from about 50 first-principles calculations, we explored this set of structures using the enumerated list and found the lowest one, shown in Fig. 6. From a physical point of view, the final answer is not too surprising. The cell bears a strong resemblance to the $L1_1$ structure of Cu–Pt.

Guided by the cluster expansion predictions, we used direct first principles calculations to check larger unit cells, including that shown in Fig. 6. Both first principles and cluster expansion indicate that, in contradiction to Ref. [19], there are *no* structures at 15:17

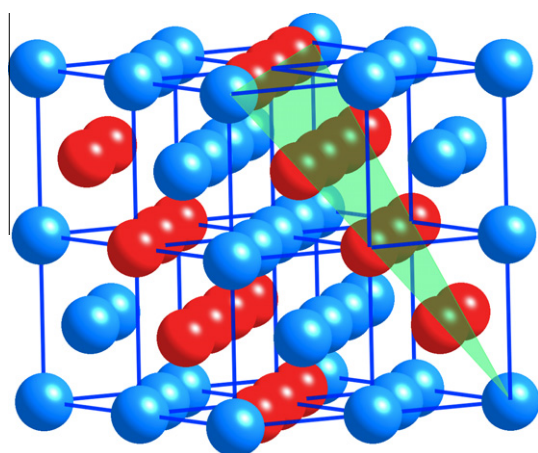


Fig. 6. The lowest-energy 32-atom Ag–Pt structure with 15:17 stoichiometry. Note that this structure is similar to the prototype CuPt structure ($L1_1$), which has alternating (1 1 1) planes of pure Pt and pure Ag. The only difference between $L1_1$ and the structure shown here is the presence of Pt “substitutional defects” in the Ag planes (indicated by the green plane), shown on the corners of this $2 \times 2 \times 2$ supercell.

stoichiometry that are stable with respect to a two-phase mixture of the $L1_1$ structure and a Pt-rich solid solution. Further evidence that $L1_1$ is the stable phase is given by the fact that all the low-lying 15:17 structure predicted by cluster expansion are essentially the $L1_1$ structure with anti-site defects (e.g., take note of the $L1_1$ -like nature of the structure in Fig. 6).

One might suspect that the structure in Fig. 6 is stabilized by vibrational entropy, but our extensive calculations of the phonon density of states reveal that our original finding ($L1_1$ is stable, 15:17 is not) is *strengthened* by including vibrational entropy effects. Furthermore, a symmetry-leveraged method[10] combined with GULP calculations found the same lowest-energy structure at 15:17 and the same result— $L1_1$ is a ground state, the structure at 15:17 is not. A full exposition of our Ag–Pt follow-up study will be discussed in another publication.

5. Summary

Our previous algorithm[16,17] enumerated derivative superstructures over the entire composition range. Because the total number of structures is a rapidly increasing function of cell size and number of atom types, there are practical limitations to its application. For example, in fcc- or bcc-based binary systems, it was impractical to go beyond cell sizes of about 28 atoms/cell. In ternary systems the limit is about 16 atoms/cell. Although these sizes are sufficient for many cases, there are other cases where the ability to enumerate larger cell sizes and in a narrow concentration range would be useful.

We developed an extension to the original algorithm that can limit the enumeration to a single composition.[20] This can be used to explore a limited concentration range in much larger unit cells. In our example of Ag–Pt, we enumerated all configurations for a fixed stoichiometry of 15:17. Cases like Ag–Pt where the *composition* of an unknown phase is known but other structural information is not are not uncommon in the alloy phase diagram literature and this algorithm is especially useful in these cases. Another common case where one may want to enumerate much larger cells (but where the total numbers of unique configurations is not impractically large) is the case of the dilute limit. With the new algorithm, for cubic or hexagonal binary alloys, one could easily explore unit cell sizes up to 64 or so if the composition is below 5% or 6%.

In the original algorithm, a key component for efficiency was the construction of a hash table and hash function, a common construct in computer science. In a list of all possible atomic configurations, the hash function mapped *configurations* onto consecutive integers. In this extension of the original algorithm, a new hash function is developed that maps *permutations* of atomic configurations (for a fixed concentration) onto consecutive integers. This extension is incorporated into the original code which is available for download [21].

Appendix A

A.1. Including site restrictions

As mentioned in the introduction, the user may wish to impose *site restrictions* as well as *composition restrictions*. Cases arise where the materials problem at hand dictates that some atom types will only occupy a subset of the sites. One obvious solution to joint site-composition restrictions is to generate the list of all configurations that satisfy the concentration restrictions and then, in a post-processing step, delete those that violate the *site* restrictions. However, the unrestricted list may be prohibitively long or take too much cpu time to generate. The site restrictions may substantially

reduce the size of the list—even by orders of magnitude—and we wish to take advantage of this reduction a priori. How can one generate a list of configurations simultaneously subject to both site and composition restrictions?

In the absence of *site restrictions*, all valid configurations can be generated simply by iterating over all the integers between 1 and C , where C is the total number of combinations, as given by the multinomial in Eq. 1. The one-to-one mapping between the integers and configurations (using the algorithm discussed in Section 3) enables efficient “crossing out” of symmetrically-equivalent configurations because it constitutes a minimal hash table and a perfect hash function.

In the case of *both* site restrictions and specified multiplicities (composition restrictions), we will continue to use the same hash table (no longer minimal) and hash function (still perfect) but require a more efficient way to generate the configurations that satisfy the restrictions. We adapt a tree search to generate all legal configurations.

The example shown in Fig. 7 illustrates the approach. We enumerate all ternary colorings of four sites, subject to the site restrictions that red cannot appear on site 4, green cannot appear on site 1, blue cannot appear on site 3. The composition restriction requires that red always appear once or twice, green no more than once, and blue no more than twice. The composition and site restrictions can be given as a set of “masking matrices”, as shown in the figure.

We use a standard backtracking algorithm to search the tree of site-restricted colorings to find those which also satisfy the multiplicity requirements. The efficiency of the tree search is realized by skipping entire groups of colorings that are disallowed. For example, the top branch at the first level of the tree violates the site restriction that green is not allowed on site 1. The backtracking algorithm thus skips *all subsequent* sub-branches.

Similarly, composition restrictions are enforced by checking the total number of each color at each node of the tree. When a violation is encountered, all subsequent sub-branches will also be in violation and can be skipped. After the tree is searched, we index the list of remaining, viable colorings by using our hash function.

A.2. Counting the multiset permutations

We may sometimes wish to *count* the number of configurations with the given multiplicities, when it simply is not practical to list them all or when checking the implementation of the algorithm.

In other words, we would like to be able to count the number of functions from the lattice quotient group to the set of available colors, subject to a restricted set of multiplicities with equivalence induced by the full set of symmetries. Note that if we were only interested in the symmetries induced by translation—i.e., by the quotient group itself—there is a very elegant solution due to Rutherford [22] using Pólya’s Theorem. Because we include the orthonormal symmetries, our problem is uglier and not easily summarized in a formula.

It is still a Pólya problem, however. Given the full symmetry group of the lattice and superlattice, expressing those as permutations of the quotient group, we may proceed as follows: Our goal is to find the average number of colorings, with the given multiplicities, which are fixed by elements of the group. Since a coloring is fixed by a group element if and only if the cycles of the permutation are monochromatic, we must, for each group element, determine how many colorings with the given multiplicities are monochromatic on the cycles of the permutation.

First, we find the cycle structures of the permutations and collect them into groups with similar structures—so the following computation need be done only once for each *distinct* cycle structure.

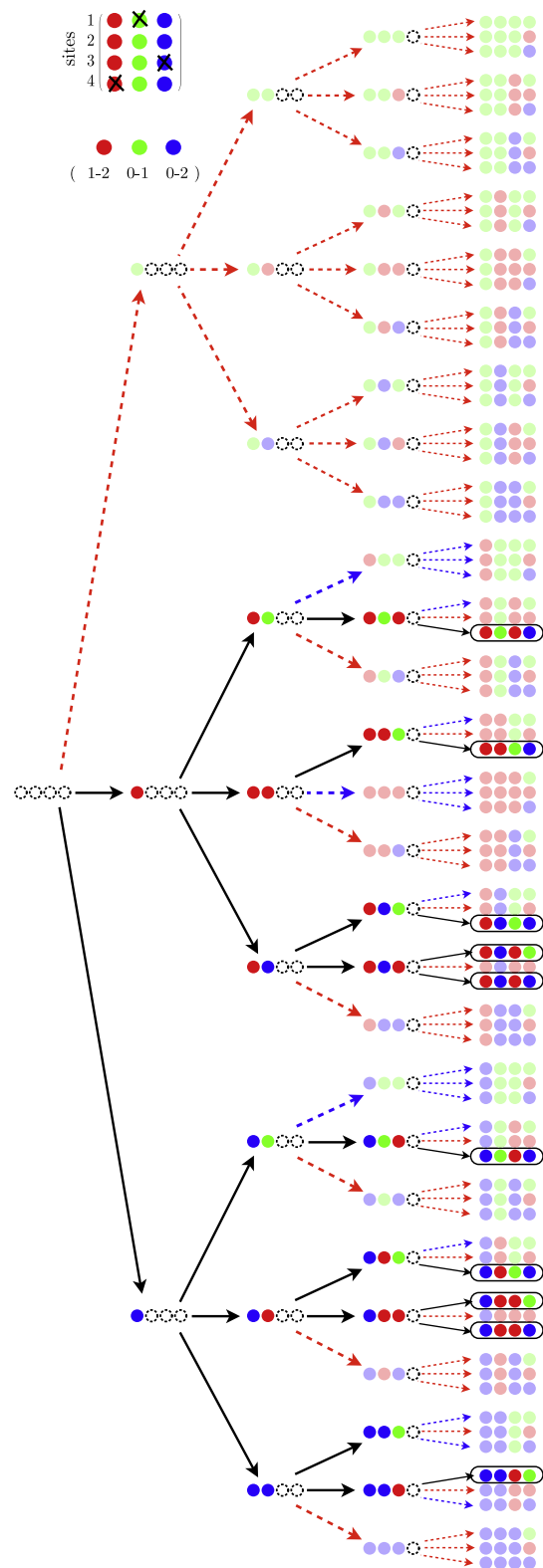


Fig. 7. Tree diagram of a sample enumeration of configurations. Branches that are “pruned” because they violate site restrictions are indicated by dashed red arrows. Branches that are eliminated because they violate composition restrictions are shown by dashed blue arrows. Legal colorings generated by the backtracking algorithm (9 out of $3^4 = 81$) are circled.

Now suppose a permutation ρ has cycles of length $c_1 < c_2 < \dots < c_t$ with multiplicities m_1, m_2, \dots, m_t respectively (so $\sum m_i c_i = n$, the order of our quotient group). Suppose also that we wish to use colors $1, 2, \dots, k$ with multiplicities a_1, a_2, \dots, a_k

respectively (so $\sum a_i = n$). Then the number of colorings with the given color-multiplicities, which are fixed by ρ , is

$$\sum_{\mathbb{M} \in \omega(\rho)} \prod_{i=1}^t \binom{c_i}{\mathbb{M}_{i1}, \mathbb{M}_{i2}, \dots, \mathbb{M}_{ik}}$$

where $\omega(\rho)$ is the set of all $t \times k$ matrices \mathbb{M} of non-negative integers satisfying the following simultaneous conditions

$$(*) \quad \sum_{j=1}^k \mathbb{M}_{ij} = m_i, \quad \forall i,$$

$$(**) \quad \sum_{i=1}^t c_i \mathbb{M}_{ij} = a_j, \quad \forall j.$$

Finding the matrices in $\omega(\rho)$ is an easy “brute force” problem because the entries in \mathbb{M} can be easily and severely bounded by the requirements of equations (*) and (**). Note that \mathbb{M}_{ij} represents how many of the cycles of length c_i are colored with color j .

Using Maple, we applied this counting algorithm to many small problems where we could compare the answer to our enumeration algorithm. In all such cases, it instantly gave the correct answer. Then, merely to test its speed, we applied the counting algorithm to a superlattice of size 48 over the FCC parent lattice. The answer, a nineteen-digit decimal number, required only five minutes in Maple—but was obviously beyond the reach of any enumeration algorithm.

References

- [1] S.M. Woodley, R. Catlow, Nature Materials 7 (2008) 937.
- [2] J.C. Schön, M. Jansen, Zeitschrift für Kristallographie 216 (2001) 307.
- [3] C. Mellot-Draznieks, S. Girard, G. Férey, J.C. Schön, Z. Cancarevic, M. Jansen, Chemistry – A European Journal 8 (2002) 4102. ISSN 1521-3765.
- [4] D.J. Wales, J.P.K. Doye, The Journal of Physical Chemistry A 101 (1997) 5111. <<http://pubs.acs.org/doi/pdf/10.1021/jp970984n>>.
- [5] R.L. Johnston, Dalton Transactions (2003) 4193–4207.
- [6] N.L. Abraham, M.I.J. Probert, Physical Review B 73 (2006) 224104.
- [7] A.R. Oganov, C.W. Glass, The Journal of Chemical Physics 124 (2006) 244704.
- [8] J. Pannetier, J. Bassas-Alsina, J. Rodriguez-Carvajal, V. Caignaert, Nature 346 (1990) 343.
- [9] A. Fadda, G. Fadda, Physical Review B 82 (2010) 104105.
- [10] Bryce Meredig and Chris Wolverton recently developed a genetic algorithm-based crystal structure prediction approach that leverages incomplete symmetry information provided by X-ray analysis <<http://meetings.aps.org/link/BAPS.2011.MAR.W32.2>>.
- [11] A. van de Walle, G. Ceder, Journal of Phase Equilibria 23 (2002) 348.
- [12] A. van de Walle, M. Asta, G. Ceder, CALPHAD 26 (2002) 539.
- [13] L.G. Ferreira, S.-H. Wei, A. Zunger, The International Journal of Supercomputer Applications 5 (1991) 34.
- [14] N.A. Zarkevich, T.L. Tan, D.D. Johnson, Physical Review B (Condensed Matter and Materials Physics) 75 (2007) 104203, pages 12.
- [15] A body of work by M. Hosoya is related to our enumeration problem. Hosoya's algorithm enumerates all possible crystal structures of any parent lattice of which derivative structures are a subset. Though this approach is not useful for the normal applications of derivative structures, interested readers may wish to consult Ref. [23] and its preceding works.
- [16] G.L.W. Hart, R.W. Forcade, Physical Review B 77 (2008) 224115.
- [17] G.L.W. Hart, R.W. Forcade, Phys. Rev. B 80 (2009) 014120.
- [18] The positions in the sequence can be mapped directly back to their atomic positions in the three-dimensional lattice using the Hermite Normal Form matrix that defines the superlattice.
- [19] P. Durussel, P. Feschotte, Journal of Alloys and Compounds 239 (1996) 226.
- [20] Configurations for composition ranges can be enumerated by applying the algorithm repeatedly to several concentrations. The enumeration code <<http://sourceforge.net>> includes this functionality [21].
- [21] A FORTRAN95 implementation of the original algorithm of Refs. [16,17], incorporating the extensions discussed in this paper <<http://sourceforge.net/projects/enum/>>.
- [22] J.S. Rutherford, Acta Crystallographica Section A 51 (1995) 672.
- [23] M. Hosoya, Bulletin of the College of Science, University of the Ryukyus 44 (1987) 11.