Journal of Computational Physics 228 (2009) 4858–4880

Contents lists available at ScienceDirect

Journal of Computational Physics

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

Efficient algorithms for discrete lattice calculations

M. Arndt [a], V. Sorkin [b], E.B. Tadmor [b,*]

[a] *School of Mathematics, University of Minnesota, Minneapolis, MN 55455, USA*
[b] *Department of Aerospace Engineering and Mechanics, University of Minnesota, Minneapolis, MN 55455, USA*

A R T I C L E   I N F O

Article history:
Received 29 September 2008
Received in revised form 25 March 2009
Accepted 27 March 2009
Available online 7 April 2009

PACS:
02.60.Cb
61.50.Ah

Keywords:
Lattice algorithms
Lattice reduction
Nearest neighbor
Cluster computation
Quasicontinuum

A B S T R A C T

We discuss algorithms for lattice-based computations, in particular lattice reduction, the detection of nearest neighbors, and the computation of clusters of nearest neighbors. We focus on algorithms that are most efficient for low spatial dimensions (typically $d = 2, 3$) and input data within a reasonably limited range. This makes them most useful for physically oriented numerical simulations, for example of crystalline solids. Different solution strategies are discussed, formulated as algorithms, and numerically evaluated.

1. Introduction

Scientific computing often involves lattices and algorithms operating on them. For example, the atoms in a crystalline solid are arranged in the form of a lattice, and numerical codes that simulate the behavior of such crystals need to perform many operations on this lattice. This includes the identification of all atoms within a given radius of a lattice site for energy and force calculations and the location of the nearest lattice site to an arbitrary point. In particular, the current work was motivated by lattice algorithms needed for the quasicontinuum (QC) method [13–15]. Lattices are also used in other physical applications, such as the Ising model [9], lattice Monte Carlo [7], lattice protein folding algorithms [16], as well as in more abstract settings, such as integer linear programming problems in operations research, lattice-based cryptography and communication theory [1].

Lattice algorithms and their computational complexity have been intensively studied from the algebraic point of view. This resulted in the development of highly-sophisticated algorithms that have good scaling properties as the spatial dimension d gets large or the lattice gets highly distorted. For an overview of such lattice algorithm, see, *e.g.*, [1,6]. A more general classical reference for lattices and their properties is [2]. However, these sophisticated algorithms are not always optimal for many applications that are oriented more towards physics or engineering, where the spatial dimension d is typically low (mostly $d = 2, 3$), and lattice distortions are typically limited to a physically-relevant range so that the scaling properties

* Corresponding author. Tel.: +1 612 625 6642.
E-mail address: tadmor@aem.umn.edu (E.B. Tadmor).

of the algorithms do not come into play. In these cases, simpler approaches that are significantly easier to implement can be superior to the more sophisticated techniques.

This paper describes lattice algorithms that are tailored to practical physical applications in numerical simulation and evaluates their performance on a set of test problems. The authors feel that there is a gap between the mathematically complex algorithms mentioned above and the naive "brute-force" approaches that are often used in practical applications. The purpose of this paper is to fill this gap for certain common lattice problems and to make algorithms readily-available for application.

We deal with three problems: lattice reduction, detection of nearest lattice sites, and computation of clusters of nearest neighbors. As noted above, this study was motivated by the QC method, nevertheless, these problems have been formulated in a general way to make them applicable and useful for a large variety of applications. We consider both simple lattices and multilattices, also called "lattices with a basis", which correspond to a set of inter-penetrating simple lattices.

A key starting point for many algorithms is a suitable lattice reduction that determines an optimal set of lattice vectors that can considerably accelerate subsequent operations. In Section 2, we discuss two variants of lattice reduction: the classical LLL reduction [8] that provides approximate, globally-optimized lattice vectors and a pairwise reduction approach that results in lattice vectors that are pairwise optimal but not necessarily globally optimal. The advantages and disadvantages of these approaches are discussed. Numerical studies of their performance appear in subsequent sections where they are incorporated into other algorithms.

Section 3 deals with the detection of nearest lattice sites, also known as the closest vector problem, for both simple lattices and multilattices. We discuss two strategies: a naive brute-force algorithm and a new approach which we refer to as the *short-list algorithm*. The latter is based on investing some computation time in advance to determine a small set of candidate lattice sites that is subsequently used to speed up the actual process of neighbor detection. This is advantageous when multiple lattice sites need to be detected for the same lattice structure. Both algorithms are evaluated numerically.

In Section 4, we discuss the computation of clusters of nearest neighbors, *i.e.*, determination of the set of all lattice sites within a given radius of a specified lattice site for both simple lattices and multilattices. This can be seen as a variation of the closest vector problem in which only the single nearest lattice site is detected. We discuss and numerically assess the performance of two strategies that we developed: the *shell algorithm* and the *on-the-fly algorithm*.

Concluding remarks are given in Section 5. The appendix contains proofs and technical details omitted from the main text in order not to interrupt the flow of reading.

2. Lattice reduction

A *simple lattice* $\mathcal{L}$ in d-dimensional space is an infinite discrete set of points that are integer linear combinations of lattice vectors, $\boldsymbol{a}_i \in \mathbb{R}^d$, for $i = 1, \dots, d$,[1]

$$\mathcal{L} = \left\{ \sum_{i=1}^{d} \boldsymbol{a}_i n_i : n_i \in \mathbb{Z} \right\} = \{\boldsymbol{A}\boldsymbol{n} : \boldsymbol{n} \in \mathbb{Z}^d\}. \tag{1}$$

In the second term, we subsume the lattice vectors as the column vectors of the matrix $\boldsymbol{A} \in \mathbb{R}^{d\times d}$, *i.e.*, $\boldsymbol{A} = (\boldsymbol{a}_1 \cdots \boldsymbol{a}_d)$. To avoid degenerate lattices, we require the lattice vectors $\boldsymbol{a}_i$ to be linearly independent, or equivalently the matrix $\boldsymbol{A}$ to be invertible.

The lattice definition in (1) is not unique because the lattice vectors are not uniquely determined. Two lattices spanned by $\boldsymbol{A}$ and $\bar{\boldsymbol{A}}$ coincide,

$$\{\boldsymbol{A}\boldsymbol{n} : \boldsymbol{n} \in \mathbb{Z}^d\} = \{\bar{\boldsymbol{A}}\boldsymbol{n} : \boldsymbol{n} \in \mathbb{Z}^d\}, \tag{2}$$

if and only if the matrix $\boldsymbol{M} = \boldsymbol{A}^{-1}\bar{\boldsymbol{A}}$ is *unimodular*, *i.e.*, $\boldsymbol{M}$ and its inverse $\boldsymbol{M}^{-1} = \bar{\boldsymbol{A}}^{-1}\boldsymbol{A}$ are integer matrices, or equivalently if $\boldsymbol{M}$ is an integer matrix with determinant ± 1, *i.e.*, $|\det \boldsymbol{M}| = 1$.

As an example, consider the sheared square lattice spanned by

$$\boldsymbol{a}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \boldsymbol{a}_2 = \begin{bmatrix} c \\ 1 \end{bmatrix} \tag{3}$$

for some $c \in \mathbb{R}$. The same lattice is also spanned by

$$\bar{\boldsymbol{a}}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \bar{\boldsymbol{a}}_2 = \begin{bmatrix} 2+c \\ 1 \end{bmatrix}, \tag{4}$$

as shown in Fig. 1. We have

$$\boldsymbol{A} = \begin{bmatrix} 1 & c \\ 0 & 1 \end{bmatrix}, \quad \bar{\boldsymbol{A}} = \begin{bmatrix} 1 & 2+c \\ 0 & 1 \end{bmatrix}. \tag{5}$$

[1] $\{\boldsymbol{A}\boldsymbol{n} : \boldsymbol{n} \in \mathbb{Z}^d\}$ denotes the set of all vectors of the form $\boldsymbol{A}\boldsymbol{n}$ that is restricted by the colon notation to all vectors $\boldsymbol{A}\boldsymbol{n}$ for which $\boldsymbol{n}$ is an integer vector, $\boldsymbol{n} \in \mathbb{Z}^d$. A similar notation is used throughout the paper. Note that bold uppercase letters denote matrices, and that bold lowercase letters denote vectors.

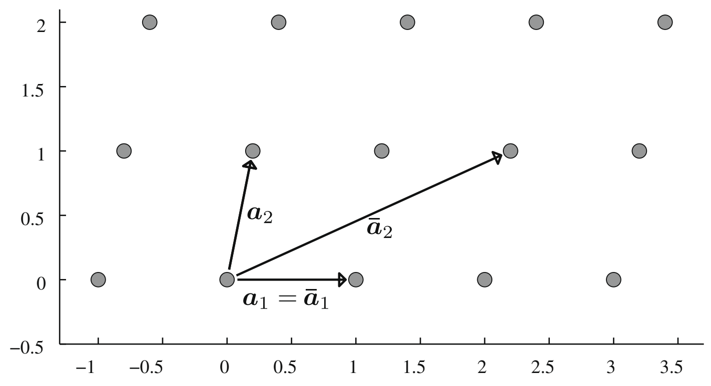


Fig. 1. Freedom in choosing lattice vectors. The lattice vectors $(\boldsymbol{a}_1, \boldsymbol{a}_2)$ and $(\bar{\boldsymbol{a}}_1, \bar{\boldsymbol{a}}_2)$ both span this two-dimensional lattice.

Here we see that

$$\boldsymbol{M} = \boldsymbol{A}^{-1}\bar{\boldsymbol{A}} = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}, \quad \boldsymbol{M}^{-1} = \bar{\boldsymbol{A}}^{-1}\boldsymbol{A} = \begin{bmatrix} 1 & -2 \\ 0 & 1 \end{bmatrix}, \tag{6}$$

are both integer matrices, and that $\det \boldsymbol{M} = 1$.

The freedom in choosing different lattice vectors for the same lattice can be used to speed up certain lattice algorithms such as finding nearest lattice sites (Section 3) or constructing clusters of lattice sites (Section 4). The process of obtaining "optimal" lattice vectors for a given lattice is called *lattice reduction*. Different lattice reduction algorithms adopt different objectives for optimizing the lattice vectors. A common objective is minimizing the length of the lattice vectors. Another one is keeping the angles between pairs of lattice vectors as close to 90° as possible.

A prominent algorithm for lattice reduction is the LLL algorithm developed by Lenstra et al. [8]. It is based on a Gram–Schmidt orthogonalization of the lattice vectors where the coefficients are rounded to an integer so that the spanned lattice does not change. Additionally, vectors are swapped during the orthogonalization process. Upon completion of the LLL algorithm, the reduced lattice vectors fulfill the condition

$$\prod_{i=1}^{d} \|\boldsymbol{a}_i\| \leqslant 2^{d(d-1)/4} |\det \boldsymbol{A}|, \tag{7}$$

where $\|\boldsymbol{x}\| = \sqrt{\sum_{i=1}^{d} x_i^2}$ denotes the Euclidean norm of the vector $\boldsymbol{x} \in \mathbb{R}^d$. The determinant of $\boldsymbol{A}$ is equal to the volume of the lattice unit cell and is independent of the specific lattice vectors. Note that for any lattice we have

$$|\det \boldsymbol{A}| \leqslant \prod_{i=1}^{d} \|\boldsymbol{a}_i\| \tag{8}$$

by the Hadamard inequality with equality satisfied if and only if the lattice vectors are orthogonal. Property (7) states that the volume deviates from the product of the lattice vector lengths only by a constant factor and therefore constitutes a certain measure of orthogonality of the LLL-reduced lattice vectors.

The LLL algorithm achieves a global lattice reduction, *i.e.*, the reduced lattice fulfills the optimality condition (7) which involves all lattice vectors at the same time. However, it does not in general produce lattice vectors that are pairwise shortest or whose angles are pairwise closest to 90°. In the following, we derive a lattice reduction algorithm based on pairwise length reduction and show that it is equivalent to a pairwise reduction based on orthogonality. We thus refer to this approach as the *pairwise reduction* (PW) algorithm.

Consider the lattice vectors $\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_d$. The corresponding lattice remains unchanged if we replace $\boldsymbol{a}_i$ by $\boldsymbol{a}_i - m\boldsymbol{a}_j$ where m is an integer and $i \neq j$, as this is a change of lattice vectors of the type described in (2), where $\boldsymbol{M} = \boldsymbol{I}$ except for $\boldsymbol{M}_{ji} = -m$. We choose m such that the length of $\|\boldsymbol{a}_i - m\boldsymbol{a}_j\|$ is minimal among all integers m. This leads to

$$m = \text{round}\left(\frac{\boldsymbol{a}_i \cdot \boldsymbol{a}_j}{\|\boldsymbol{a}_j\|^2}\right), \tag{9}$$

where the rounding is done towards the nearest integer (or, precisely speaking, any nearest integer if it is not unique). For a reason to be discussed later, we require that the rounding is always done towards zero whenever the argument is an integer plus half, *i.e.*,

$$\text{round}(x) = \text{sign}(x)\lceil |x| - \tfrac{1}{2} \rceil = \begin{cases} \lceil x - \frac{1}{2} \rceil & \text{if } x \geqslant 0, \\ \lfloor x + \frac{1}{2} \rfloor & \text{if } x < 0. \end{cases} \tag{10}$$

Here $\lceil x \rceil$ and $\lfloor x \rfloor$ denote the smallest integer that is greater than or equal to x (ceiling operation) and the largest integer that is smaller than or equal to x (floor operation), respectively. For vector arguments, $\boldsymbol{x} \in \mathbb{R}^d$, the rounding is meant component-wise,

$(\text{round}(\boldsymbol{x}))_i = \text{round}(x_i)$. It can be proved that the new vector $\boldsymbol{a}_i - m\boldsymbol{a}_j$ with m defined in (9) is shortest among all $m \in \mathbb{Z}$, and additionally that it is the vector whose angle with $\boldsymbol{a}_j$ is closest to 90°, see Lemma A.1 in the appendix. This means that our particular choice of m makes the vector both shortest and as orthogonal to $\boldsymbol{a}_j$ as possible. This technique of reducing a pair of vectors originally goes back to Gauss [5, Article 171].

We use this technique to define the PW algorithm given in Algorithm 1. In this algorithm, we make use of the fact that if some vector $\boldsymbol{a}_l$ is already length-reduced by some shorter vector $\boldsymbol{a}_s$, then the shorter vector $\boldsymbol{a}_s$ is automatically length-reduced by the longer vector $\boldsymbol{a}_l$. See Lemma A.2 in the appendix for the precise statement and the proof. This saves one half of the reduction operations.

Algorithm 1. Pairwise lattice reduction (PW).

```
Input: lattice vectors $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_d$
do
    $terminate := \text{true}$
    for $i := 1, \ldots, d-1$
        for $j := i+1, \ldots, d$
            if $\|\boldsymbol{a}_i\| \geqslant \|\boldsymbol{a}_j\|$
                $l := i; s := j$
            else
                $l := j; s := i$
            $m := \text{round}\,((\boldsymbol{a}_l \cdot \boldsymbol{a}_s)/\|\boldsymbol{a}_s\|^2)$
            if $(m \neq 0)$
                $\boldsymbol{a}_l := \boldsymbol{a}_l - m\boldsymbol{a}_s$
                $terminate := \text{false}$
while ($terminate$ = false)
Output: reduced lattice vectors $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_d$
```

Due to the special definition in (10) of the rounding function it can be shown that whenever $m\neq 0$ in some step, the length of the corresponding vector $\boldsymbol{a}_l$ is in fact reduced by a positive amount and never stays the same, which could lead to a never-ending alternating algorithm. Hence the sum $\sum_{i=1}^{d} \|\boldsymbol{a}_i\|$ is strictly decreasing as long as at least one coefficient $m\neq 0$. Because of the discreteness of the lattice, the algorithm always terminates after a finite number of steps.

Algorithm 1 concludes with a set of lattice vectors that are pairwise reduced according to both length and angle. This means that subtracting an integer multiple of one vector from another vector never reduces its length or brings the angle closer to 90°. However, it can be shown that this is not a global property, *i.e.*, subtracting two or more integer multiples at the same time could lead to an improvement. Obtaining a set of lattice vectors which is globally reduced in this sense is a considerably more difficult task, and to the best knowledge of the authors, no efficient algorithm for arbitrary space dimension d exists that does so.

It is worth noting that the technique used here is similar to the Euclidean Algorithm for finding the greatest common divisor (gcd) of two positive integers. This similarity is due to the fact that both lattice vectors and gcd share the same structure of invariance; two positive integers n_i, n_j have the same gcd as $\bar{n}_i, \bar{n}_j$ if and only if

$$(n_i \; n_j) = (\bar{n}_i \; \bar{n}_j)\boldsymbol{M} \tag{11}$$

for some unimodular matrix $\boldsymbol{M}$; compare this to (2).

Finally, an estimate of the computational complexity of the LLL algorithm was derived in [8] for the case of lattice vectors with integer components. It was shown that the running time of the algorithm is $O(d^4 \log B)$, where $B = \max_i |\boldsymbol{a}_i|^2$ is an upper bound on the magnitude squared of the lattice vectors. This analysis can be generalized to lattice vectors whose components are rational numbers (ratios of integers). Such complexity estimates could in principle be carried over to the PW algorithm. However, they seem to be of little relevance in the context of this paper since physically-relevant lattices normally do not admit lattice vectors of rational numbers. Moreover, the analysis would not be helpful even in cases where the components of the lattice vectors are rational numbers, because there is no apparent connection between the physically-relevant length and direction of the lattice vectors and the magnitude of the integer numbers involved. Small changes in the lengths or directions of the real lattice vectors can already lead to highly varying lengths of the involved integer numbers, compare, *e.g.*, 1/2 and 1,000,001/2,000,000. The parameter B appearing in the LLL bound could be very different for these two cases, whereas the real lattice vector component is almost identical. For this reason, we disregard the topic of complexity estimates here.

3. Nearest lattice site detection

One problem that frequently occurs in many applications is the determination of the lattice site that is nearest to some given point $\bar{\boldsymbol{x}} \in \mathbb{R}^d$ in space, also known as the closest vector problem. An example is mesh generation and mesh refinement within the QC method, where mesh nodes are constrained to occupy lattice sites. Whenever the initial mesh is computed or

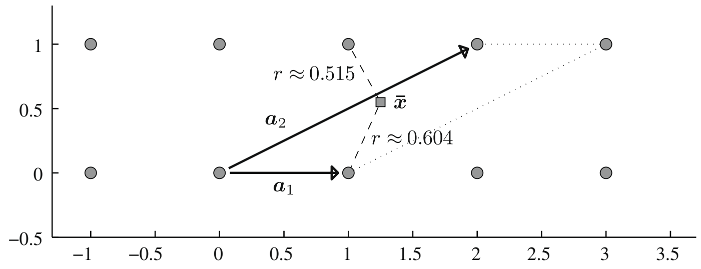


Fig. 2. Example of a point $\bar{\boldsymbol{x}}$ (square) whose nearest lattice site $\boldsymbol{a}_2 - \boldsymbol{a}_1$ is not a corner of the unit cell (vectors and dotted lines).

the mesh is refined by adding nodes during automatic mesh refinement, the nodes must be moved to the nearest lattice site. The fast determination of nearest lattice sites is therefore essential for an efficient QC implementation.

We begin with a simple lattice $\mathcal{L} = \{\boldsymbol{Am} : \boldsymbol{m} \in \mathbb{Z}^d\}$. Assume we are given an arbitrary point $\bar{\boldsymbol{x}} = \boldsymbol{A}\bar{\boldsymbol{n}} \in \mathbb{R}^d$. Here $\bar{\boldsymbol{n}} \in \mathbb{R}^d$, but in general $\bar{\boldsymbol{n}} \notin \mathbb{Z}^d$ because $\bar{\boldsymbol{x}}$ need not be a lattice site. The problem of detecting the nearest lattice site is to find a lattice site $\boldsymbol{x} = \boldsymbol{An} \in \mathcal{L}$ that is nearest to $\bar{\boldsymbol{x}}$, *i.e.*,[2]

$$\boldsymbol{x} = \arg\min_{\boldsymbol{x}' \in \mathcal{L}} \|\boldsymbol{x}' - \bar{\boldsymbol{x}}\|. \tag{12}$$

Equivalently, the problem can be stated as finding $\boldsymbol{n} \in \mathbb{Z}^d$ such that

$$\|\boldsymbol{An} - \bar{\boldsymbol{x}}\| \leqslant \|\boldsymbol{Am} - \bar{\boldsymbol{x}}\| \quad \text{for all } \boldsymbol{m} \in \mathbb{Z}^d. \tag{13}$$

The nearest lattice site problem becomes trivial if the lattice is spanned by an orthogonal matrix $\boldsymbol{A}$, *i.e.*, $\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}$, since in this case

$$\|\boldsymbol{An} - \bar{\boldsymbol{x}}\| = \|\boldsymbol{n} - \boldsymbol{A}^{-1}\bar{\boldsymbol{x}}\|, \tag{14}$$

and $\boldsymbol{n}$ is obtained by solving $\bar{\boldsymbol{n}} = \boldsymbol{A}^{-1}\bar{\boldsymbol{x}}$ for $\bar{\boldsymbol{n}}$ and then rounding each component of $\bar{\boldsymbol{n}}$ to the nearest integer, $\boldsymbol{n} = \operatorname{round}(\bar{\boldsymbol{n}})$, where round( ) is defined in (10). In particular, the nearest lattice site is always one of the 2^d corners of the unit cell containing $\bar{\boldsymbol{x}}$. This property holds as well if the lattice vectors are only orthogonal, $\boldsymbol{a}_i \cdot \boldsymbol{a}_j = 0$ for $i \neq j$, but not necessarily normalized, $\|\boldsymbol{a}_i\| = 1$.

For non-orthogonal lattice vectors, $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_d$, this is in general no longer true as can be seen from the following example, see also Fig. 2. Let

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}, \quad \bar{\boldsymbol{x}} = \begin{bmatrix} 1.25 \\ 0.55 \end{bmatrix}, \quad \bar{\boldsymbol{n}} = \begin{bmatrix} 0.15 \\ 0.55 \end{bmatrix}. \tag{15}$$

The four corners $\boldsymbol{m} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, and $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ of the unit cell containing $\bar{\boldsymbol{x}}$ satisfy $\|\boldsymbol{Am} - \bar{\boldsymbol{x}}\| > 0.60415$, but the nearest lattice site, $\boldsymbol{An} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ with $\boldsymbol{n} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$, has $\|\boldsymbol{An} - \bar{\boldsymbol{x}}\| \approx 0.51479$. Thus, the nearest lattice site is not a corner of the unit cell containing $\bar{\boldsymbol{x}}$, and $\boldsymbol{n}$ cannot be determined simply by rounding.

Note however that the matrix $\boldsymbol{A}$ from (15) can be reduced to an orthogonal matrix by the lattice reduction techniques discussed in Section 2, which makes the nearest lattice site detection trivial for this particular example. This does not hold for lattices generated by arbitrary matrices $\boldsymbol{A}$, but hints that reducing $\boldsymbol{A}$ to some matrix that is as orthogonal as possible can significantly reduce the computational complexity. Later in this section, we numerically compare the performance of lattice site detection for unmodified, pairwise reduced, and LLL-reduced lattice vector matrices $\boldsymbol{A}$.

Let us note that for dimension $d = 2$, it can be proved that the nearest lattice site is always one of the four corners of the surrounding unit cell if the matrix $\boldsymbol{A}$ is PW-reduced. In this case, no further strategy beyond length-reducing the matrix and then checking which one of the four corners is nearest is necessary. For higher dimension $d \geqslant 3$, this is no longer true as can be shown by counterexamples, and other strategies become necessary.

3.1. Brute-force method

Given any matrix $\boldsymbol{A}$, either reduced in some sense or not, the naive brute-force approach to determine the nearest lattice site is to compute the distances to all lattice sites in the vicinity of $\bar{\boldsymbol{x}}$ and to take the minimal one. This vicinity needs to be large enough to ensure that the nearest lattice site is not missed, but should be as small as possible for computational efficiency.

[2] The arg min operator returns the value of the argument for which the specified expression is minimum. Thus, $\arg\min_x f(x)$ returns the value of x at which $f(x)$ is minimum.

In practice, one identifies a corner of the unit cell that contains $\bar{\boldsymbol{x}}$, $\mathrm{round}(\bar{\boldsymbol{n}}) = \mathrm{round}(\boldsymbol{A}^{-1}\bar{\boldsymbol{x}})$, and then scans a rectangular set of indices $\boldsymbol{n}$ around this corner, *i.e.*, all $\boldsymbol{n}$ with

$$|n_i - \mathrm{round}(\bar{n}_i)| \leqslant R_b(\boldsymbol{A}) \tag{16}$$

for some bound $R_b(\boldsymbol{A})$. The bound $R_b(\boldsymbol{A})$ can be determined as follows:

$$\begin{aligned}\|\boldsymbol{n} - \mathrm{round}(\bar{\boldsymbol{n}})\|_\infty &\leqslant \|\boldsymbol{n} - \bar{\boldsymbol{n}}\|_\infty + \|\bar{\boldsymbol{n}} - \mathrm{round}(\bar{\boldsymbol{n}})\|_\infty \leqslant \|\boldsymbol{A}^{-1}\|_{\infty,2}\|\boldsymbol{A}(\boldsymbol{n} - \bar{\boldsymbol{n}})\| + \|\bar{\boldsymbol{n}} - \mathrm{round}(\bar{\boldsymbol{n}})\|_\infty \\ &= \min_{\boldsymbol{m}\in\mathbb{Z}^d} \|\boldsymbol{A}^{-1}\|_{\infty,2}\|\boldsymbol{A}(\boldsymbol{m} - \bar{\boldsymbol{n}})\| + \|\bar{\boldsymbol{n}} - \mathrm{round}(\bar{\boldsymbol{n}})\|_\infty \\ &\leqslant \min_{\boldsymbol{m}\in\mathbb{Z}^d} \|\boldsymbol{A}^{-1}\|_{\infty,2}\|\boldsymbol{A}\|_{2,\infty}\|(\boldsymbol{m} - \bar{\boldsymbol{n}})\|_\infty + \|\bar{\boldsymbol{n}} - \mathrm{round}(\bar{\boldsymbol{n}})\|_\infty \leqslant \frac{1}{2}\|\boldsymbol{A}^{-1}\|_{\infty,2}\|\boldsymbol{A}\|_{2,\infty} + \frac{1}{2},\end{aligned} \tag{17}$$

where $\|\boldsymbol{n}\|_\infty = \max_i |n_i|$ is the maximum norm and $\|\boldsymbol{A}\|_{\infty,2}$ and $\|\boldsymbol{A}\|_{2,\infty}$ are operator norms. Note that the operator norms are defined as the smallest numbers that satisfy

$$\|\boldsymbol{Av}\|_\infty \leqslant \|\boldsymbol{A}\|_{\infty,2}\|\boldsymbol{v}\| \quad \text{and} \quad \|\boldsymbol{Av}\| \leqslant \|\boldsymbol{A}\|_{2,\infty}\|\boldsymbol{v}\|_\infty \tag{18}$$

for all vectors $\boldsymbol{v}$. We have

$$\|\boldsymbol{A}\|_{\infty,2} = \max_{i=1,\dots,d} \sqrt{\sum_{j=1}^{d} A_{ij}^2}, \quad \|\boldsymbol{A}\|_{2,\infty} \leqslant \sqrt{\sum_{i=1}^{d}\left(\sum_{j=1}^{d}|A_{ij}|\right)^2}. \tag{19}$$

There is no easily computable formula for the exact value of $\|\boldsymbol{A}\|_{2,\infty}$ as there is for $\|\boldsymbol{A}\|_{\infty,2}$, but the upper estimate given here is sufficient for our application.

Estimate (17) states that the nearest lattice site $\boldsymbol{An}$ fulfills the condition

$$\mathrm{round}(\bar{n}_i) - \lfloor R_b(\boldsymbol{A})\rfloor \leqslant n_i \leqslant \mathrm{round}(\bar{n}_i) + \lfloor R_b(\boldsymbol{A})\rfloor \tag{20}$$

for all components n_i of the vector $\boldsymbol{n}$, where

$$R_b(\boldsymbol{A}) := \frac{1}{2}\left(\|\boldsymbol{A}^{-1}\|_{\infty,2}\|\boldsymbol{A}\|_{2,\infty} + 1\right). \tag{21}$$

We define the brute-force search range

$$\mathcal{R}_{bf} := \left\{\boldsymbol{m} \in \mathbb{Z}^d : -\lfloor R_b(\boldsymbol{A})\rfloor \leqslant m_i \leqslant \lfloor R_b(\boldsymbol{A})\rfloor\right\}. \tag{22}$$

Then the nearest lattice site $\boldsymbol{An}$ is determined by computing the distances $\|\boldsymbol{An} - \bar{\boldsymbol{x}}\|$ for all $\boldsymbol{n} = \mathrm{round}(\bar{\boldsymbol{n}}) + \boldsymbol{m}$ with $\boldsymbol{m} \in \mathcal{R}_{bf}$ and choosing the one with minimal distance. The corresponding brute-force algorithm is shown in Algorithm 2.

Algorithm 2. Algorithm to detect the nearest lattice site. For the search range, $\mathcal{R}$, we use either $\mathcal{R} = \mathcal{R}_{bf}$ (brute-force method) or $\mathcal{R} = \mathcal{R}_{sl}$ (short-list method).

```
Input: matrix $A$, point $\bar{\boldsymbol{x}}$, search range $\mathcal{R}$
$\bar{\boldsymbol{n}} := \boldsymbol{A}^{-1}\bar{\boldsymbol{x}}$
find minimum of $\|\boldsymbol{An} - \bar{\boldsymbol{x}}\|$ among all $\boldsymbol{n} = \mathrm{round}(\bar{\boldsymbol{n}}) + \boldsymbol{m}$ with $\boldsymbol{m} \in \mathcal{R}$
Output: $n$
```

For the 2D lattice given in (15), for example, we have $R_b(\boldsymbol{A}) = \frac{1}{2}(\sqrt{5}\sqrt{10} + 1) \approx 4.036$, so $\lfloor R_b(\boldsymbol{A})\rfloor = 4$. Thus, the brute-force algorithm would have to scan $9 \times 9 = 81$ lattice sites.

3.2. Short-list method

The brute-force approach described in the previous section is reasonable if one only needs to compute the nearest lattice sites for a few points $\bar{\boldsymbol{x}}$. In many applications, though, this needs to be done for a great many points $\bar{\boldsymbol{x}}$, so it is advisable to invest some time in a preprocessing step in order to compute a smaller search range. This search range needs to be computed only once for a given lattice, hence one saves computation time when determining many lattice sites later. We refer to the new approach that uses the smaller search range as the *short-list* algorithm.

Consider the lattice site detection technique in Algorithm 2. This algorithm works with indices $\boldsymbol{m} = \boldsymbol{n} - \mathrm{round}(\bar{\boldsymbol{n}})$. Thus, $\mathrm{round}(\bar{\boldsymbol{n}})$ is subtracted from all indices $\boldsymbol{n}$ that are potential candidates for the nearest lattice site. Observe that $\bar{\boldsymbol{x}} - \boldsymbol{A}\,\mathrm{round}(\bar{\boldsymbol{n}}) \in \boldsymbol{A}[-\frac{1}{2},\frac{1}{2}]^d$ by construction.[3] Hence, the subtraction shifts the search to lattice sites around the origin. The

[3] Here $[a,b]^d$ and $\{a,b\}^d$ denote the set of d-vectors with components within the range from a to b or equal to a or b, respectively. Hence, $\boldsymbol{A}[-\frac{1}{2},\frac{1}{2}]^d$ is the set of all points from the unit cell centered around the origin, and $\boldsymbol{A}\{-\frac{1}{2},\frac{1}{2}\}^d$ is the set of all corners of this unit cell.

key of the short-list method is to find a search range $\mathcal{R}_{sl}$ that contains all lattice sites that are closest to at least one point in $\boldsymbol{A}\left[-\frac{1}{2},\frac{1}{2}\right]^d$. This range does not depend on the actual point $\bar{\boldsymbol{x}}$ and can thus be computed in advance for all possible points $\bar{\boldsymbol{x}}$. Once $\mathcal{R}_{sl}$ is computed, Algorithm 2 with $\mathcal{R} = \mathcal{R}_{sl}$ can be applied to any point $\bar{\boldsymbol{x}}$.

The construction of $\mathcal{R}_{sl}$ is based on Voronoi cells and their representation using half-spaces. Careful estimates of the involved terms lead to[4]

$$\mathcal{R}_{sl} = \left\{\boldsymbol{m} \in \mathcal{R}_{bf} : \forall \boldsymbol{m}' \in \mathcal{R}_{bf}, \boldsymbol{m}' \neq \boldsymbol{m} : \boldsymbol{A}\{-\tfrac{1}{2},\tfrac{1}{2}\}^d \cap \mathcal{H}(\boldsymbol{m},\boldsymbol{m}') \neq \emptyset\right\}, \tag{23}$$

where the half-space $\mathcal{H}(\boldsymbol{m},\boldsymbol{m}')$ is defined by

$$\mathcal{H}(\boldsymbol{m},\boldsymbol{m}') := \left\{\boldsymbol{x} \in \mathbb{R}^d : (\boldsymbol{x} - \tfrac{1}{2}\boldsymbol{A}(\boldsymbol{m}+\boldsymbol{m}')) \cdot \boldsymbol{A}(\boldsymbol{m}'-\boldsymbol{m}) \leqslant 0\right\}. \tag{24}$$

The detailed description of the derivation of $\mathcal{R}_{sl}$ is lengthy. The interested reader is referred to Appendix A.2 for the details. Algorithm 3 shows how to compute $\mathcal{R}_{sl}$ as a preprocessing step.

Algorithm 3. Preprocessing step to determine the search range for the short-list method, $\mathcal{R}_{sl}$. The command `break` terminates the innermost loop.

```
Input: $\boldsymbol{A}$
$\mathcal{R}_{sl} := \emptyset$
compute $R_b(\boldsymbol{A})$ from (21) and $\mathcal{R}_{bf}$ from (22)
for all $\boldsymbol{m} \in \mathcal{R}_{bf}$
    $intersect$ :=$true$
    for all $\boldsymbol{m}' \in \mathcal{R}_{bf}, \boldsymbol{m}' \neq \boldsymbol{m}$
        $corner\_inside$ := false
        for all $\boldsymbol{c} \in \{-\frac{1}{2},\frac{1}{2}\}^d$
            if $(\boldsymbol{c} - \frac{1}{2}(\boldsymbol{m}'+\boldsymbol{m}))^T \boldsymbol{A}^T \boldsymbol{A}(\boldsymbol{m}'-\boldsymbol{m}) \leqslant 0$
                $corner\_inside$ := true
                break
        if $corner\_inside$ = false
            $intersect$ :=false
            break
    if $intersect$ = true
        $\mathcal{R}_{sl} := \mathcal{R}_{sl} \cup \{\boldsymbol{m}\}$
Output: $\mathcal{R}_{sl}$
```

Since $\mathcal{R}_{sl}$ is much smaller than the search range of the brute-force approach, $\mathcal{R}_{bf}$, determining $\boldsymbol{n}$ becomes much faster. For example, for the 2D lattice in (15), $\mathcal{R}_{sl}$ contains 13 lattice sites, significantly less than the 81 lattice sites needed for the brute-force algorithm. In higher dimensions, $d \geqslant 3$, the advantage of the short-list method over the brute-force method becomes even more significant.

3.3. Performance evaluation

To evaluate the performance of the algorithms discussed so far for $d = 3$, we measured running times for all six possible combinations for a sample of 100 randomly-generated matrices $\boldsymbol{A}$. In each case, the matrix can be left as is, reduced by the length reduction algorithm, or reduced by the LLL algorithm. Then, the nearest lattice site can be either determined by the brute-force or short-list algorithms. To avoid excessive running times for the non-reduced matrix, we only considered matrices $\boldsymbol{A}$ for which $R_b(\boldsymbol{A}) \leqslant 20$.

Table 1 shows running times for the different components of the respective algorithms. Here the reduction step refers to PW reduction, LLL reduction or no reduction at all for the matrix $\boldsymbol{A}$. The setup step comprises the computation of the bound $R_b(\boldsymbol{A})$ and of the search ranges $\mathcal{R}_{bf}$ and $\mathcal{R}_{sl}$ for the brute-force method and for the short-list method, respectively. The find step refers to the detection the nearest lattice site $\boldsymbol{n}$ for a single point $\bar{\boldsymbol{x}}$. The results show that the reduction step, either PW or LLL, costs nearly no computation time, but that it greatly reduces the time for the subsequent setup and find operations. Bearing in mind that the reduction step just needs to be done once for each matrix, it is a clear must. The setup phase of the short-list method costs additional time because it involves about $2^d(2R_b(\boldsymbol{A})+1)^{2d}$ vector operations, compared with only

[4] $\forall$ and $\exists$ are the universal quantifier ("for all") and the existential quantifier ("there exists"), respectively. In words, $\mathcal{R}_{sl}$ is the set of all $\boldsymbol{m}$ from $\mathcal{R}_{bf}$ such that for all $\boldsymbol{m}'$ from $\mathcal{R}_{bf}$ except for $\boldsymbol{m}$, there exists a corner $\boldsymbol{x}$ that is contained in the half-space $\mathcal{H}(\boldsymbol{m},\boldsymbol{m}')$.

Table 1
Nearest lattice site detection in a simple lattice. Results are for 100 random matrices $\boldsymbol{A} \in \mathbb{R}^{3\times3}$ with $R_b(\boldsymbol{A}) \leq 20$ and 100,000 random find operations per matrix. Running times for the individual steps are averages over all tests. The reduction and setup times are performed once for each matrix, so the average is over 100 matrices. The find time is averaged over the $100 \times 100{,}000 = 10{,}000{,}000$ find operations that were performed. The list size is an average over the 100 matrices. For the short-list method with PW reduction or LLL reduction and for the matrices considered here, the list size is always 17. The total running time is the time for the full simulation including all steps for all matrices and lattice site detections. Computations were performed on an Intel Core2 Q6600 2.40 GHz CPU.

Method	Reduction	Reduction (μs)	Setup (μs)	Find (μs)	Average list size	Total (s)
Brute-force	None	N/A	91.624	66.675	5105.272	666.756
	PW	0.171	3.107	2.647	183.192	26.472
	LLL	0.301	3.115	2.590	183.745	25.902
Short-list	None	N/A	302184.812	0.390	24.543	34.121
	PW	0.171	2594.212	0.285	17.000	3.113
	LLL	0.301	750.859	0.278	17.000	2.857

$(2R_b(\boldsymbol{A}) + 1)^d$ vector operations for the brute-force method. However, it accelerates the subsequent find operations by a factor of 9 on average, thus it pays off if many find operations are performed. Based on Table 1, the actual crossover point for which the short-list algorithm is favorable to the brute-force algorithm is 1098 find operations if PW reduction is used and 324 find operations if LLL reduction is used. The lattice-reduced approaches also have significant storage advantages over the corresponding unreduced approaches as shown by the list size requirements (*i.e.*, the average number of lattice sites scanned during a find operation).

Table 1 also shows the total running time for a practical application consisting of 100 different matrices $\boldsymbol{A} \in \mathbb{R}^{3\times3}$ with 100,000 find operations each. Clearly the short-list method for the reduced matrices is best, with LLL slightly faster than PW. The advantage of the PW approach is that it has a clear geometric significance and is much simpler to implement than LLL, while being nearly as efficient. The brute-force methods and the non-reduced methods are not competitive.

An important variation of the nearest lattice site detection problem is the restriction to a finite-sized domain, $\Omega \subset \mathbb{R}^d$. Consider the problem of seeking the lattice site from $\mathcal{L} \cap \Omega$ that is nearest to $\bar{\boldsymbol{x}} \in \Omega$. If $\bar{\boldsymbol{x}}$ is more than $\frac{1}{2}\|\boldsymbol{A}\|_{2,\infty}$ away from the boundary $\partial\Omega$ of the domain, *i.e.*, $\mathrm{dist}(\bar{\boldsymbol{x}}, \partial\Omega) > \frac{1}{2}\|\boldsymbol{A}\|_{2,\infty}$, the nearest lattice site to $\bar{\boldsymbol{x}}$ from the infinite lattice $\mathcal{L}$ is contained in Ω, as can be seen by an estimate similar to (17). Thus the problem does not differ from finding the nearest lattice site from the full lattice $\mathcal{L}$, and both the short-list method and the brute-force method will find the correct solution. However, within distance $\frac{1}{2}\|\boldsymbol{A}\|_{2,\infty}$ to the boundary, the nearest site from $\mathcal{L} \cap \Omega$ might not be the nearest site from $\mathcal{L}$, and the carefully constructed search list for the original problem might miss the correct lattice site.

It is therefore advisable to proceed as follows. First, the nearest lattice site from the full lattice $\mathcal{L}$ is determined using the short-list algorithm. If this lattice site lies within the domain Ω, we have found the solution. Otherwise, we fall back to searching from a much larger set, such as the one used in the brute-force method. It is important to note, though, that the correct result is not guaranteed even for the brute-force method if the domain Ω is highly irregular. In the worst case, any site from $\mathcal{L} \cap \Omega$ could be nearest.

In practice, it will not be necessary to revert to the slow brute-force approach too often if the domain Ω is well-behaved. Consider, for example, the case of a convex domain Ω. Then the set of points $\bar{\boldsymbol{x}} \in \Omega$ for which the short-list method possibly fails has volume $\leqslant \frac{1}{2}\|\boldsymbol{A}\|_{2,\infty}|\partial\Omega|$, where $|\partial\Omega|$ denotes the surface area of Ω. Thus, the ratio of possibly problematic points to all points in Ω is at most $\frac{1}{2}\|\boldsymbol{A}\|_{2,\infty}|\partial\Omega|/|\Omega|$, where $|\Omega|$ denotes the volume of Ω. This fraction is small if Ω is sufficiently large, hence we still have a quite efficient method.

3.4. Nearest lattice site detection for multilattices

Up to now, we discussed how to detect the nearest lattice site within a simple lattice as given by (1). We now extend this to multilattices.

A *multilattice* is a set of S inter-penetrating simple lattices called *sublattices*.[5] The set of lattice sites making up the multilattice is

$$\mathcal{L} = \{\boldsymbol{A}(\boldsymbol{n} + \boldsymbol{\nu}_\alpha) : \boldsymbol{n} \in \mathbb{Z}^d, \alpha = 1, \ldots, S\}, \tag{25}$$

where[3] $\boldsymbol{\nu}_\alpha \in [0,1)^d (\alpha = 1, \ldots, S)$ are the positions of the sublattices (also called *fractional position vectors*) relative to the nominal simple lattice that serves as the scaffolding for the multilattice (sometimes called the "skeletal lattice"). We say that each nominal lattice site has S sublattice sites associated with it. Clearly, if $S = 1$, the multilattice reduces to a simple lattice, possibly shifted relative to the nominal lattice.

[5] It is also possible to think of a multilattice structure as a simple lattice with more than one point associated with each lattice site. This interpretation has led to the term *lattice with a basis*, which is common in the physics literature. In this context, the set of "points" located at each lattice site is called the *basis*. For a crystal structure, the basis corresponds to a set of atoms. If a single atom is placed at each lattice site, the result is a simple crystal structure. If the basis contains multiple atoms, the result is a complex or multilattice crystal.

When applying the nearest lattice site algorithm to the multilattice in (25), it is necessary to account for the fractional position vectors $\boldsymbol{v}_\alpha$. The objective is to find the lattice site $\boldsymbol{A}(\boldsymbol{n}+\boldsymbol{v}_\alpha)$ satisfying

$$\|\boldsymbol{A}(\boldsymbol{n}+\boldsymbol{v}_\alpha)-\bar{\boldsymbol{x}}\| \leqslant \|\boldsymbol{A}(\boldsymbol{m}+\boldsymbol{v}_\beta)-\bar{\boldsymbol{x}}\| \quad \text{for all } \boldsymbol{m}\in\mathbb{Z}^d,\ \beta=1,\ldots,S. \tag{26}$$

Compare this condition to (13). Obviously

$$\boldsymbol{A}(\boldsymbol{n}+\boldsymbol{v}_\alpha)-\bar{\boldsymbol{x}}=\boldsymbol{A}\boldsymbol{n}-(\bar{\boldsymbol{x}}-\boldsymbol{A}\boldsymbol{v}_\alpha). \tag{27}$$

Thus, for a single position vector ($S=1$), the problem reduces to the case of a simple lattice where the point $\bar{\boldsymbol{x}}$ is replaced by $\bar{\boldsymbol{x}}-\boldsymbol{A}\boldsymbol{v}_1$. For a multilattice with several sublattices, we apply one of the techniques for simple lattices discussed earlier to each position vector $\boldsymbol{v}_\alpha$, individually. In this manner, we obtain lattice sites $\boldsymbol{A}(\boldsymbol{n}_\alpha+\boldsymbol{v}_\alpha)$ that are nearest to $\bar{\boldsymbol{x}}$ among the shifted simple lattice $\{\boldsymbol{A}\boldsymbol{m}+\boldsymbol{v}_\alpha : \boldsymbol{m}\in\mathbb{Z}^d\}$. The overall solution to (26) is then obtained by choosing the lattice site $\boldsymbol{A}(\boldsymbol{n}_\alpha+\boldsymbol{v}_\alpha)$ with the minimal distance to $\bar{\boldsymbol{x}}$ among all the sublattices $\alpha=1,\ldots,S$.

Note that we have not addressed the important issue of essential versus non-essential descriptions of multilattices [11]. The *essential* description is the multilattice with the minimal value of S that reproduces the lattice structure. Of course, it is always possible to represent a simple lattice or a multilattice with S sublattices by another multilattice with larger periodicity and correspondingly larger S. This is a *non-essential* description. The algorithms given above will work for non-essential descriptions, but would be accelerated by using the essential representation. The problem of determining the essential description for a given multilattice is an important problem that lies outside the scope of this paper. See for example [10] for a discussion of this issue.

4. Cluster construction

Another important task is to determine all lattice sites that lie within a certain radius r around the origin or a specified lattice site in a simple lattice or multilattice that have undergone a uniform deformation. The application we encounter most often is the computation of atomic interactions under a short-range potential.[6] In this case, r corresponds to the cutoff radius r_{cut} of the interaction potential. Since this is the application we have in mind, we will refer to r as the cutoff radius and to the sphere of radius r as the cutoff sphere.

We consider a multilattice with S inter-penetrating sublattices that are spanned by the column vectors $\boldsymbol{a}_1,\boldsymbol{a}_2,\ldots,\boldsymbol{a}_d\in\mathbb{R}^d$ of the matrix $\boldsymbol{A}\in\mathbb{R}^{d\times d}$ and which have fractional position vectors $\boldsymbol{v}_\alpha$ for $\alpha=1,\ldots,S$,

$$\mathcal{L}_{ref}=\{\boldsymbol{A}(\boldsymbol{n}+\boldsymbol{v}_\alpha):\boldsymbol{n}\in\mathbb{Z}^d,\alpha=1,\ldots,S\}. \tag{28}$$

The lattice $\mathcal{L}_{ref}$ is referred to as the *reference configuration*. The sublattices undergo a uniform deformation described by the affine mapping $\boldsymbol{y}(\boldsymbol{x})=\boldsymbol{F}\boldsymbol{x}+\boldsymbol{u}_\alpha$, where $\boldsymbol{F}\in\mathbb{R}^{d\times d}$ is a regular matrix and where $\boldsymbol{u}_\alpha\in\mathbb{R}^d$ are *shift vectors* that displace the sublattices relative to each other. In continuum mechanics applications, $\boldsymbol{F}$ is the *deformation gradient*, which is how we will refer to it here. The *deformed configuration* $\mathcal{L}$ of the multilattice follows from (28) as

$$\mathcal{L}=\{\boldsymbol{F}\boldsymbol{A}(\boldsymbol{n}+\boldsymbol{v}_\alpha)+\boldsymbol{u}_\alpha:\boldsymbol{n}\in\mathbb{Z}^d,\alpha=1,\ldots,S\}=\{\boldsymbol{F}\boldsymbol{A}(\boldsymbol{n}+\boldsymbol{s}_\alpha):\boldsymbol{n}\in\mathbb{Z}^d,\alpha=1,\ldots,S\}=\{\boldsymbol{F}\boldsymbol{A}\boldsymbol{n}+\boldsymbol{\delta}_\alpha:\boldsymbol{n}\in\mathbb{Z}^d,\alpha=1,\ldots,S\}, \tag{29}$$

where $\boldsymbol{s}_\alpha=\boldsymbol{v}_\alpha+\boldsymbol{A}^{-1}\boldsymbol{F}^{-1}\boldsymbol{u}_\alpha$ and where we refer to $\boldsymbol{\delta}_\alpha=\boldsymbol{F}\boldsymbol{A}\boldsymbol{v}_\alpha+\boldsymbol{u}_\alpha$ as the *offset vector* of sublattice α.

Two problems are of interest in this context. The first problem is to find all lattice sites $(\boldsymbol{n},\alpha)$ satisfying

$$\|\boldsymbol{F}\boldsymbol{A}(\boldsymbol{n}+\boldsymbol{s}_\alpha)\|\leqslant r, \tag{30}$$

i.e., all lattice sites within distance r of the origin. The second problem is to find all lattice sites $(\boldsymbol{n},\alpha)$ that are within distance r of a given lattice site $(\bar{\boldsymbol{n}},\bar{\alpha})$,

$$\|\boldsymbol{F}\boldsymbol{A}(\boldsymbol{n}+\boldsymbol{s}_\alpha)-\boldsymbol{F}\boldsymbol{A}(\bar{\boldsymbol{n}}+\boldsymbol{s}_{\bar{\alpha}})\|\leqslant r. \tag{31}$$

Due to the translation invariance of the infinite lattice, we may set $\bar{\boldsymbol{n}}=0$ without loss of generality. Thus, the second problem becomes equivalent to the first one with the sublattice positions replaced by their differences. We therefore have the generic problem,

$$\|\boldsymbol{F}\boldsymbol{A}(\boldsymbol{n}+\boldsymbol{s})\|\leqslant r, \tag{32}$$

where $\boldsymbol{s}$ is either $\boldsymbol{s}_\alpha$ or $\boldsymbol{s}_\alpha-\boldsymbol{s}_{\bar{\alpha}}$. Below, we develop two different algorithms to efficiently determine all $\boldsymbol{n}\in\mathbb{Z}^d$ satisfying (32) for given $r>0$ and $\boldsymbol{s}\in\mathbb{R}^d$. The two problems described above are then solved by repeatedly applying one of these algorithms to all $\boldsymbol{s}=\boldsymbol{s}_\alpha$ with $\alpha=1,\ldots,S$ (first problem) or to all $\boldsymbol{s}=\boldsymbol{s}_\alpha-\boldsymbol{s}_{\bar{\alpha}}$ with $\alpha,\bar{\alpha}=1,\ldots,S$ (second problem), respectively. We derive the methods for multilattices. The case for simple lattices follows as a special case by setting $S=1$.

4.1. Algorithms for cluster construction in multilattices

In this section, we derive two methods to solve problem (32): the shell algorithm and the on-the-fly algorithm.

[6] The application we have in mind is the calculation of the constitutive response of a multilattice crystalline material using Cauchy–Born kinematics. See, *e.g.*, [4,13–15] for more details.

4.1.1. Shell algorithm

The shell algorithm is based on iterating over a *reference crystallite* $\mathcal{C}$ that is computed in advance and consists of all lattice sites in the reference configuration of the nominal lattice lying within a radius R_c of the origin (to be specified later),

$$\mathcal{C} = \{\boldsymbol{An} : \boldsymbol{n} \in \mathbb{Z}^d, \|\boldsymbol{An}\| \leqslant R_c\}. \tag{33}$$

The reference crystallite is arranged into shells, where the lattice sites in each shell have the same distance to the origin. Sorting the shells by this distance allows for an efficient iteration over the lattice sites later.

The setup of the reference crystallite is described in Algorithm 4. The algorithm is passed $\boldsymbol{A}, R_c$ and ϵ, a tolerance setting the shell thickness. Normally, ϵ will be related to the shell spacing and machine tolerance. In our calculations, which were performed in double precision, we used $\epsilon = 10^{-9}$. Algorithm 4 is self-explanatory, except for the initial selection of a pool of lattice sites $\mathcal{P}$ from which the reference crystallite is built.

The pool $\mathcal{P}$ needs to be sufficiently large to ensure that it contains all indices $\boldsymbol{n} \in \mathbb{Z}^d$ with $\boldsymbol{An} \in \mathcal{C}$, *i.e.*, $\|\boldsymbol{An}\| \leqslant R_c$. We have

$$\|\boldsymbol{An}\|^2 = \boldsymbol{n}^T\boldsymbol{A}^T\boldsymbol{An} \geqslant \mu_{\min}\|\boldsymbol{n}\|^2, \tag{34}$$

where $\mu_{\min}$ is the minimum eigenvalue of $\boldsymbol{A}^T\boldsymbol{A}$. Thus

$$\|\boldsymbol{n}\| \leqslant \frac{\|\boldsymbol{An}\|}{\sqrt{\mu_{\min}}} \leqslant \frac{R_c}{\sqrt{\mu_{\min}}}. \tag{35}$$

Because $\|\boldsymbol{n}\|_\infty \leqslant \|\boldsymbol{n}\|$, we get the conservative bound $\|\boldsymbol{n}\|_\infty \leqslant \lfloor R_c/\sqrt{\mu_{\min}}\rfloor$. This leads to the easily computable set

$$\mathcal{P} = \left\{\boldsymbol{n} \in \mathbb{Z}^d : \|\boldsymbol{n}\|_\infty \leqslant \lfloor R_c/\sqrt{\mu_{\min}}\rfloor\right\}. \tag{36}$$

The reference crystallite only has to be generated once for a given nominal lattice. Normally this does not change during a simulation. Once generated, it can be used to obtain the list of lattice sites lying within the cutoff radius r of a lattice site in the deformed multilattice in the following manner.

Algorithm 4. Shell algorithm: Construction of the reference crystallite, $\mathcal{C}$.

```
  Input: $\boldsymbol{A}, R_c, \epsilon$
  compute $\mu_{\min}$, the minimum eigenvalue of $\boldsymbol{A}^T\boldsymbol{A}$
  $\mathcal{P} := \{\boldsymbol{n} \in \mathbb{Z}^d : \|\boldsymbol{n}\|_\infty \leqslant \lfloor R_c/\sqrt{\mu_{\min}}\rfloor\}$
  $\mathcal{C} := \emptyset$
  $N_c := 0$
  for all $\boldsymbol{n} \in \mathcal{P}$
    if $\|\boldsymbol{An}\| \leqslant R_c$
      $\mathcal{C} := \mathcal{C} \cup \{\boldsymbol{An}\}$
      $N_c := N_c + 1$
    sort $\mathcal{C}$ in ascending order based on magnitude. Result: $\|\mathcal{C}(1)\| \leqslant \cdots \leqslant \|\mathcal{C}(N_c)\|$
  $R_s(1) := \|\mathcal{C}(1)\|$
  $\mathcal{S}(1) := \emptyset$
  $shell := 1$
  for $\boldsymbol{i} := 1, \ldots, N_c$
    $R := \|\mathcal{C}(\boldsymbol{i})\|$
    if $|R - R_s(shell)| < \epsilon$
      $\mathcal{S}(shell) := \mathcal{S}(shell) \cup \{\mathcal{C}(\boldsymbol{i})\}$
    else
      $shell := shell + 1$
      $R_s(shell) := R$
      $\mathcal{S}(shell) := \{\mathcal{C}(\boldsymbol{i})\}$
Output: shells $S(j)$ with radii $R_s(j)$
```

Algorithm 5. Shell algorithm: Cluster construction.

```
Input: $\boldsymbol{F}, \boldsymbol{\delta}, r$, shells $S(j)$ with radii $R_s(j)$ from Algorithm 4
compute $R_i(\boldsymbol{F}, \boldsymbol{\delta}, r)$ from (39)
$j := 1$
while $R_s(j) \leqslant R_i(\boldsymbol{F}, \boldsymbol{\delta}, r)$

  for all $\boldsymbol{x} \in S(j)$
    if $\|\boldsymbol{Fx} + \boldsymbol{\delta}\| \leqslant r$
      output $\boldsymbol{Fx} + \boldsymbol{\delta}$
  $j := j + 1$
```

The basic idea is that given a deformation gradient $\boldsymbol{F}$ and offset vector $\boldsymbol{\delta}$, it is possible to map r back to an *influence radius* $R_i(\boldsymbol{F}, \boldsymbol{\delta}, r)$ in the reference configuration [14,15]. The influence radius is the distance of the nominal lattice site furthest from the origin (in the reference configuration) that has at least one associated sublattice site that lies inside the cutoff sphere in the deformed configuration. Since the reference crystallite is sorted into shells, it is straightforward to locate all lattice sites within the cutoff sphere by only looping over shells with a radius R_i or smaller. The lattice sites associated with the nominal lattice sites in each shell are mapped to the deformed configuration by applying the deformation gradient and adding on the offset vector for a given sublattice and retaining those that lie inside the cutoff sphere. The precise procedure is given in Algorithm 5. The calculation of the influence radius R_i is discussed next.

For a simple lattice, a sphere of lattice sites in the deformed configuration is mapped back to an ellipsoid in the reference configuration by $\boldsymbol{F}^{-1}$. In a multilattice, each sublattice is mapped back to an ellipsoid with the center offset by $-\boldsymbol{F}^{-1}\boldsymbol{\delta}_\alpha(\alpha = 1, \dots, S)$ due to the sublattice offsets as illustrated in Fig. 3. Our objective is to compute the influence radius $R_i(\boldsymbol{F}, \boldsymbol{\delta}_{\max}, r)$, which is the distance from the origin to the furthest point on any of the ellipsoids. Thus, R_i is the distance of the nominal lattice point An that is furthest from the origin and that satisfies the condition $\|\boldsymbol{FAn} + \boldsymbol{\delta}_\alpha\| \leqslant r$ for at least one $\alpha \in \{1, \dots, S\}$.

We obtain a bound for this problem as follows. We have

$$\lambda_{\min}\|\boldsymbol{An}\|^2 \leqslant \boldsymbol{n}^T\boldsymbol{A}^T\boldsymbol{F}^T\boldsymbol{FAn} = \|\boldsymbol{FAn}\|^2, \tag{37}$$

where $\lambda_{\min} = \lambda_{\min}(\boldsymbol{F}^T\boldsymbol{F})$ denotes the smallest eigenvalue of $\boldsymbol{F}^T\boldsymbol{F}$. The bound for multilattices is then obtained by adding on the sublattice offset,

$$\|\boldsymbol{An}\| \leqslant \frac{\|\boldsymbol{FAn}\|}{\sqrt{\lambda_{\min}}} \leqslant \frac{\|\boldsymbol{FAn} + \boldsymbol{\delta}_\alpha\| + \|\boldsymbol{\delta}_\alpha\|}{\sqrt{\lambda_{\min}}} \leqslant \frac{r + \|\boldsymbol{\delta}_\alpha\|}{\sqrt{\lambda_{\min}}}. \tag{38}$$

Hence we choose

$$R_i(\boldsymbol{F}, \boldsymbol{\delta}_{\max}, r) := \frac{r + \|\boldsymbol{\delta}_{\max}\|}{\sqrt{\lambda_{\min}(\boldsymbol{F}^T\boldsymbol{F})}}, \tag{39}$$

where $\boldsymbol{\delta}_{\max}$ is the maximal offset vector,

$$\boldsymbol{\delta}_{\max} := \boldsymbol{\delta}_\beta \quad \text{such that } \|\boldsymbol{\delta}_\beta\| \geqslant \|\boldsymbol{\delta}_\alpha\| \quad \text{for all } \alpha \in \{1, \dots, S\}. \tag{40}$$

Note that this condition has already been given in [15] without rigorously proving its validity.

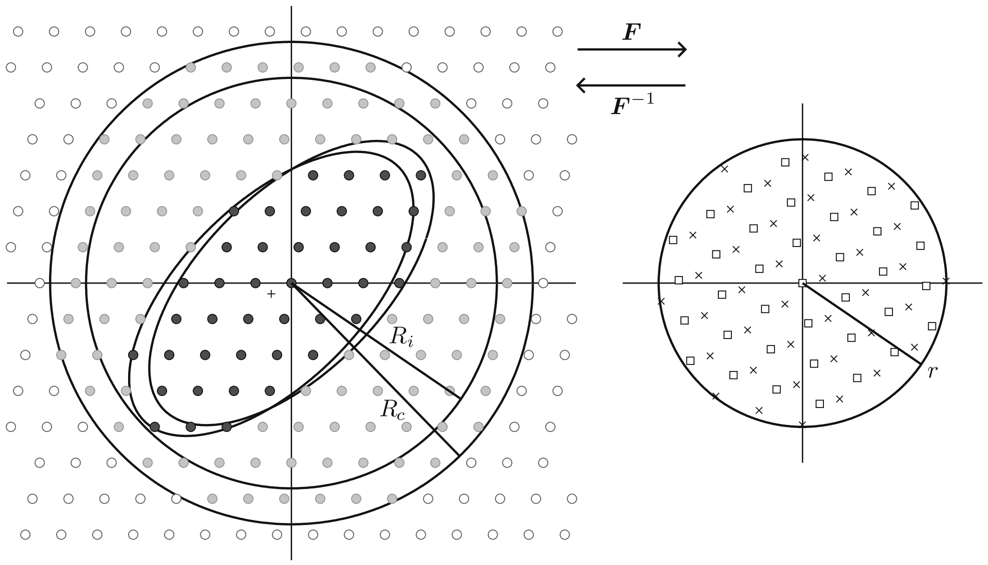


Fig. 3. Influence radius for a two-dimensional multilattice with two sublattices. The right part of the figure shows all lattice sites within distance r of the origin in the deformed configuration. One sublattice lies on the deformed nominal lattice sites ($\Box$) and the other sublattice ($\times$) is displaced by an offset vector $\boldsymbol{\delta}$. The two sublattices are mapped back by $\boldsymbol{F}^{-1}$ to two ellipses in the reference configuration (shown on the left). The ellipse associated with the $\Box$ sublattice is centered on the origin, and the other associated with the $\times$ sublattice is shifted by $-\boldsymbol{F}^{-1}\boldsymbol{\delta}$ relative to the origin. The center of the offset ellipse is marked with a + symbol. The circles on the left frame correspond to the nominal lattice sites. All nominal lattice sites falling inside the ellipses (black) have sublattice sites lying inside the cutoff sphere. The influence radius R_i computed for this deformation is shown. The figure also shows the outer radius R_c of the stored crystallite. Any deformation gradient with $R_i \leq R_c$ can be computed. Lattice sites inside the crystallite are shown in gray and lattice sites outside it are white.

The radius R_c of the reference crystallite needs to be chosen sufficiently large so that $R_c \geqslant R_i(\boldsymbol{F}, \boldsymbol{\delta}_{\max}, r)$ for all deformation gradients to be applied. Alternatively, R_c can be set to a sensible physical value, which can be used to constrain the behavior of a solver. For example, during an energy minimization in a QC simulation, excessive moves along a search direction can be detected by the condition $R_c < R_i(\boldsymbol{F}, \boldsymbol{\delta}_{\max}, r)$, alerting the solver that it needs to back up to more physically-reasonable values.

Algorithm 6. Shell algorithm: Cluster construction with PW-lattice reduction. Alternatively, LLL reduction can be used instead of the pairwise algorithm.

```
Input: $\boldsymbol{A}, \boldsymbol{F}$
Apply lattice reduction Algorithm 1 to $\boldsymbol{A}$. Result:$\bar{\boldsymbol{A}}$
Build crystallite, $\mathcal{C}$, from Algorithm 4 with $\bar{\boldsymbol{A}}$ instead of $\boldsymbol{A}$
For each deformation gradient $\boldsymbol{F}$:
    $\boldsymbol{B} := \boldsymbol{F}\bar{\boldsymbol{A}}$
    Apply lattice reduction Algorithm 1 to $\boldsymbol{B}$. Result: $\bar{\boldsymbol{B}}$
    $\bar{\boldsymbol{F}} := \bar{\boldsymbol{B}}\bar{\boldsymbol{A}}^{-1}$
    Apply the shell method in Algorithm 5 with $\bar{\boldsymbol{F}}$ instead of $\boldsymbol{F}$.
```

Regarding the computational complexity of the shell algorithm, volume considerations show that the number of lattice points that satisfy the generic problem (32) is about $N_{\text{gen}} = r^d/|\det(\boldsymbol{FA})|$ for large radii r. Similarly, the number of points scanned by the shell method is about $N_{\text{scan}} = R_i^d/|\det(\boldsymbol{A})| = (r + \|\boldsymbol{\delta}_{\max}\|)^d/\left(\lambda_{\min}^{d/2}|\det(\boldsymbol{A})|\right)$. Thus, the multiplicative overhead of the shell method is

$$\frac{N_{\text{scan}}}{N_{\text{gen}}} = \frac{(r + \|\boldsymbol{\delta}_{\max}\|)^d}{r^d} \frac{|\det(\boldsymbol{F})|}{\lambda_{\min}^{d/2}}. \tag{41}$$

Here, the ratio $|\det(\boldsymbol{F})|/\lambda_{\min}^{d/2}$ measures how much the ellipsoid that contains the points $\boldsymbol{A}(\boldsymbol{n} + \boldsymbol{s})$ satisfying (32) deviates from a ball. For $d = 3$, *e.g.*, the ratio is $\sqrt{\lambda_{\text{mid}}\lambda_{\max}/\lambda_{\min}}$, where λ_{mid} and $\lambda_{\max}$ denote the middle eigenvalue and the largest eigenvalue of $\boldsymbol{F}^T\boldsymbol{F}$, respectively. Let us note that these estimates are accurate for large radii r, whereas for small r, the volume approach is imprecise due to dominating discreteness effects.

The shell method can often be accelerated by applying a lattice reduction procedure to obtain a "smaller" deformation gradient by factoring out the lattice-invariant portion of the deformation.[7] The procedure involves a setup stage that has to be done once and an application stage, which is repeated for each deformation gradient.

The setup stage involves the following steps. First, PW or LLL lattice reduction is applied to the reference lattice vectors in matrix $\boldsymbol{A}$. This results in an optimized reference lattice given by $\bar{\boldsymbol{A}}$.[8] Recall that the reference lattice spanned by $\bar{\boldsymbol{A}}$ is the same as the lattice spanned by $\boldsymbol{A}$, only the corresponding indices $\boldsymbol{n}$ may differ. Second, the crystallite $\mathcal{C}$ is computed with Algorithm 4, where $\boldsymbol{A}$ is replaced by $\bar{\boldsymbol{A}}$. This completes the setup stage.

The application stage involves the following steps. For every deformation gradient $\boldsymbol{F}$, the deformed lattice $\boldsymbol{B} = \boldsymbol{F}\bar{\boldsymbol{A}}$ is lattice reduced to $\bar{\boldsymbol{B}}$. The corresponding reduced deformation gradient $\bar{\boldsymbol{F}}$ follows from the requirement $\bar{\boldsymbol{B}} = \bar{\boldsymbol{F}}\bar{\boldsymbol{A}}$, thus

$$\bar{\boldsymbol{F}} = \bar{\boldsymbol{B}}\bar{\boldsymbol{A}}^{-1}. \tag{42}$$

The shell method in Algorithm 5 is then applied using $\bar{\boldsymbol{F}}$ instead of the original deformation gradient $\boldsymbol{F}$. The approach is outlined in Algorithm 6 with PW-lattice reduction. Of course, LLL lattice reduction could be used instead.

The saving in computational time as a result of this algorithm can be significant. For example, if a lattice-invariant shear is applied to the system, *i.e.*, a shear that leaves the lattice unchanged, then $\bar{\boldsymbol{B}} = \bar{\boldsymbol{A}}$ and therefore $\bar{\boldsymbol{F}} = \boldsymbol{I}$, which is an optimal case for the shell method. Timing studies for this example are given in Sections 4.2 and 4.3 for a simple lattice and multilattices.

4.1.2. On-the-fly algorithm

The second technique is to determine the lattice sites entirely in the deformed configuration. We refer to this method as the on-the-fly (OTF) algorithm. Rather than looping over lattice sites in the reference configuration and mapping them to the deformed configuration, the lattice vectors themselves are mapped to the deformed configuration, $\boldsymbol{FA}$, and the lattice sites lying within the cutoff sphere are determined on-the-fly by means of d nested loops with certain bounds. The bounds are chosen carefully to exactly give the correct set of lattice sites, at the cost of some computational overhead to compute these bounds.

Let

$$\boldsymbol{G} := \boldsymbol{A}^T\boldsymbol{F}^T\boldsymbol{FA} \in \mathbb{R}^{d\times d}. \tag{43}$$

[7] A *lattice invariant* deformation is a deformation that leaves the infinite lattice unchanged.

[8] The lattice vectors given in textbooks for physically-relevant crystal structures are normally already reduced. In this case the lattice reduction step would simply return $\bar{\boldsymbol{A}} = \boldsymbol{A}$.

Note that $\boldsymbol{G}$ is symmetric positive definite, provided that both $\boldsymbol{A}$ and $\boldsymbol{F}$ are regular. Then our problem is to identify all $\boldsymbol{n} \in \mathbb{Z}^d$ with

$$(\boldsymbol{n}+\boldsymbol{s})^T\boldsymbol{G}(\boldsymbol{n}+\boldsymbol{s}) \leqslant r^2. \tag{44}$$

To isolate the last component n_d of $\boldsymbol{n}$, we subdivide $\boldsymbol{G}, \boldsymbol{n}$, and $\boldsymbol{s}$ into blocks as

$$\boldsymbol{G} = \begin{bmatrix} \tilde{\boldsymbol{G}} & \boldsymbol{g} \\ \boldsymbol{g}^T & \gamma \end{bmatrix}, \quad \boldsymbol{n} = \begin{bmatrix} \tilde{\boldsymbol{n}} \\ n_d \end{bmatrix}, \quad \boldsymbol{s} = \begin{bmatrix} \tilde{\boldsymbol{s}} \\ s_d \end{bmatrix} \tag{45}$$

with $\tilde{\boldsymbol{G}} \in \mathbb{R}^{(d-1)\times(d-1)}, \boldsymbol{g} \in \mathbb{R}^{d-1}, \gamma \in \mathbb{R}, \tilde{\boldsymbol{n}} \in \mathbb{Z}^{d-1}, n_d \in \mathbb{Z}, \tilde{\boldsymbol{s}} \in \mathbb{R}^{d-1}$, and $s_d \in \mathbb{R}$. Using this notation, (44) reads as

$$(\tilde{\boldsymbol{n}}+\tilde{\boldsymbol{s}})^T\tilde{\boldsymbol{G}}(\tilde{\boldsymbol{n}}+\tilde{\boldsymbol{s}}) + 2(n_d+s_d)\boldsymbol{g}^T(\tilde{\boldsymbol{n}}+\tilde{\boldsymbol{s}}) + \gamma(n_d+s_d)^2 \leqslant r^2, \tag{46}$$

or equivalently

$$\left(\sqrt{\gamma}n_d + \frac{\boldsymbol{g}^T(\tilde{\boldsymbol{n}}+\tilde{\boldsymbol{s}}) + \gamma s_d}{\sqrt{\gamma}}\right)^2 \leqslant r^2 - (\tilde{\boldsymbol{n}}+\tilde{\boldsymbol{s}})^T\hat{\boldsymbol{G}}(\tilde{\boldsymbol{n}}+\tilde{\boldsymbol{s}}) \tag{47}$$

with

$$\hat{\boldsymbol{G}} := \tilde{\boldsymbol{G}} - \frac{\boldsymbol{g}\otimes\boldsymbol{g}}{\gamma}, \tag{48}$$

where $\boldsymbol{g}\otimes\boldsymbol{g}$ is the self dyad of $\boldsymbol{g}$, which in component form is $(\boldsymbol{g}\otimes\boldsymbol{g})_{ij} = g_i g_j$.

If we treat $\tilde{\boldsymbol{n}}$ as known for now, we obtain the condition

$$-\frac{\boldsymbol{g}^T(\tilde{\boldsymbol{n}}+\tilde{\boldsymbol{s}}) + \gamma s_d}{\gamma} - \sqrt{\frac{r^2 - (\tilde{\boldsymbol{n}}+\tilde{\boldsymbol{s}})^T\hat{\boldsymbol{G}}(\tilde{\boldsymbol{n}}+\tilde{\boldsymbol{s}})}{\gamma}} \leqslant n_d \leqslant -\frac{\boldsymbol{g}^T(\tilde{\boldsymbol{n}}+\tilde{\boldsymbol{s}}) + \gamma s_d}{\gamma} + \sqrt{\frac{r^2 - (\tilde{\boldsymbol{n}}+\tilde{\boldsymbol{s}})^T\hat{\boldsymbol{G}}(\tilde{\boldsymbol{n}}+\tilde{\boldsymbol{s}})}{\gamma}} \tag{49}$$

for the remaining coordinate n_d. This inequality gives the bounds for the innermost loop over n_d.

After having separated the component n_d, we now turn to n_{d-1}. Observe that (47) can only hold if its right-hand side is non-negative, *i.e.*,

$$(\tilde{\boldsymbol{n}}+\tilde{\boldsymbol{s}})^T\hat{\boldsymbol{G}}(\tilde{\boldsymbol{n}}+\tilde{\boldsymbol{s}}) \leqslant r^2. \tag{50}$$

Just as $\boldsymbol{G}$, the matrix $\hat{\boldsymbol{G}}$ is symmetric positive definite as well, see Lemma A.3 in the appendix. Hence condition (50) has the same structure as (44) where the dimensions of the vectors and matrices are $d-1$ instead of d and $(d-1)\times(d-1)$ instead of $d\times d$, respectively. Thus, the same strategy that was used for $\boldsymbol{G} \in \mathbb{R}^{d\times d}$ and $\boldsymbol{n} \in \mathbb{Z}^d$ can be used for $\hat{\boldsymbol{G}} \in \mathbb{R}^{(d-1)\times(d-1)}$ and $\tilde{\boldsymbol{n}} \in \mathbb{Z}^{d-1}$ to get the second innermost loop over n_{d-1}. This process is continued until we get the outermost loop over n_1.

The whole procedure is shown in Algorithm 7. Note that the ranges given for the matrices $\boldsymbol{G}^{(i)}$ correspond to the submatrices $\tilde{\boldsymbol{G}}, \boldsymbol{g}$ and γ used in the text. This has been done to emphasize that it is not necessary to actually declare variables for the submatrices, instead it is sufficient and usually faster to just access the corresponding part of $\boldsymbol{G}^{(i)}$. The same holds for $\boldsymbol{n}$ and $\boldsymbol{s}$.

Algorithm 7. On-the-fly algorithm: cluster construction.

```
Input: $\boldsymbol{A}, \boldsymbol{F}, \boldsymbol{s}, r$
$\boldsymbol{G}^{(d)} := \boldsymbol{A}^T\boldsymbol{F}^T\boldsymbol{F}\boldsymbol{A}$
for $i := d, \ldots, 2$
    $\boldsymbol{G}^{(i-1)} := G^{(i)}_{1\ldots i-1,1\ldots i-1} - G^{(i)}_{1\ldots i-1,i}G^{(i)}_{i,1\ldots i-1}/G^{(i)}_{i,i}$
$\alpha_1 := s_1$
$\beta_1 := \sqrt{r^2/G^{(1)}_{1,1}}$
for $n_1 := \lceil -\alpha_1 - \beta_1 \rceil, \ldots, \lfloor -\alpha_1 + \beta_1 \rfloor$
    ...
        $\alpha_i := G^{(i)}_{i,1\ldots i-1}(n+s)_{1\ldots i-1}/G^{(i)}_{i,i} + s_i$
        $\beta_i := \sqrt{(r^2 - (n+s)^T_{1\ldots i-1}\boldsymbol{G}^{(i-1)}(n+s)_{1\ldots i-1})/G^{(i)}_{i,i}}$
        for $n_i := \lceil -\alpha_i - \beta_i \rceil, \ldots, \lfloor -\alpha_i + \beta_i \rfloor$
            ...
                $\alpha_d := G^{(d)}_{d,1\ldots d-1}(n+s)_{1\ldots d-1}/G^{(d)}_{d,d} + s_d$
                $\beta_d := \sqrt{(r^2 - (n+s)^T_{1\ldots d-1}\boldsymbol{G}^{(d-1)}(n+s)_{1\ldots d-1})/G^{(d)}_{d,d}}$
                for $n_d := \lceil -\alpha_d - \beta_d \rceil, \ldots, \lfloor -\alpha_d + \beta_d \rfloor$
                    output $\boldsymbol{n}$
```

Since the OTF method directly obtains the correct set (32) of lattice points, it is faster than the shell method by a factor of the order of the multiplicative overhead given by (41). The involved constants are larger due to the complex computations of the bounds, though.

A minor possible improvement that we do not consider here is the effect of the coordinate system orientation and the ordering of the loops in Algorithm 7 on the performance of the OTF method. The method would be most efficient if the number of bounds calculations were minimized. This is achieved if outer loops correspond to spatial directions for which the bounds are closer together. Clearly,

$$\texttt{for}\; i := 0, \ldots, 2; \quad \texttt{for}\; j := 0, \ldots, 10; \quad \ldots \tag{51}$$

is more efficient than

$$\texttt{for}\; j := 0, \ldots, 10; \quad \texttt{for}\; i := 0, \ldots, 2; \quad \ldots \tag{52}$$

assuming that the bounds of the inner loop have to be computed each time. The optimal directions can be determined by computing the principle directions of the deformation. However the cost of doing so would probably offset any gains and was not done here.

4.2. Timing results for simple lattices

In this section timing results for simple lattices are presented to assess the efficiency of the methods discussed above: the shell method (S) given in Algorithm 5, the shell method with PW-lattice reduction (S + PW) given in Algorithm 6, and the OTF method given in Algorithm 7. The reference configuration for the calculations shown here is a three-dimensional simple cubic lattice with a normalized lattice parameter $a = 1$, so that

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{53}$$

To explore the dependence of the methods on the cutoff radius r, three different radii are tested: $r = 1.0$, 1.732, 3.126, corresponding to a 1st (1-NN), 3rd (3-NN), 9th (9-NN) neighbor model.[9] The models contain respectively 7, 27, 147 lattice sites in the reference configuration (including the lattice site at the origin). The normalized reference crystallite radius used in the shell method S is set to be $R_c = 20$, selected to be large enough for all investigated deformations.

In our timing analysis two types of deformations with deformation gradient $\boldsymbol{F}$ are applied to the reference configuration: simple shear and uniaxial stretch. This choice is motivated by the QR decomposition theorem, see, *e.g.*, [12]. According to this theorem, any matrix $\boldsymbol{F}$ can be factorized as $\boldsymbol{F} = \boldsymbol{QR}$, where the first factor, $\boldsymbol{Q}$, is an orthogonal matrix corresponding to rotation and reflection, and the second factor, $\boldsymbol{R}$, is an upper triangular matrix. Furthermore, the upper triangular matrix, $\boldsymbol{R}$, can be written as a product $\boldsymbol{R} = \boldsymbol{DS}$ of a diagonal matrix, $\boldsymbol{D}$, representing a stretch deformation, and an upper triangular matrix with unit diagonal, $\boldsymbol{S}$, representing a shear deformation. Rotations and reflections do not change the actual structure of the lattice. Thus, shear and stretch deformations are two prototypical deformations from which all other deformations can be constructed. The deformation gradients for simple shear and uniaxial stretch along the x-direction are given by

$$\boldsymbol{F} = \begin{bmatrix} 1 & \gamma & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \boldsymbol{F} = \begin{bmatrix} \lambda & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{54}$$

respectively. Here γ and λ are the dimensionless shear and stretch parameters that set the magnitude of the deformation.

We begin with the simple shear deformation. We provide timing results for the range $\gamma \in [0.0, 2.5]$ to explore how the magnitude of shear affects the performance of the methods. In order to measure CPU time accurately, the cluster construction procedure is repeated 10^6 times and the total CPU time is recorded, then CPU time per cluster is calculated. The timing results are shown in Fig. 4. The total computation times as functions of the applied shear γ are plotted for the three methods being considered (S, S + PW, OTF) and the three different model sizes (1-NN, 3-NN, 9-NN) associated with different cutoff radii r. It is striking that OTF and S + PW are insensitive to the magnitude of the applied shear for all three model sizes, while S is strongly affected by it. In OTF this is due to the fact that the shear happens to be optimally oriented for the loop ordering in the OTF algorithm (see Section 4.1.2) so that the number of bounds that are computed does not increase with γ. In S + PW, the reason is that an infinite lattice that is sheared along a crystallographic direction will periodically revert back to its original structure. The shears for which this occurs are called lattice-invariant shears as explained earlier. When lattice reduction is applied to a lattice invariant shear, the calculation time is the same as if the lattice were unsheared. This accounts for the zigzag shape of the S + PW curve; an effect that becomes more noticeable with increasing model size.

The result of the fact that S is sensitive to the magnitude of γ, while OTF and S + PW are not, is that for each model size there is a crossover point below which method S is optimal and above which OTF and S + PW are superior. Below the crossover, the overhead associated with the calculation of bounds for OTF and lattice reduction for S + PW offset the resulting gain of checking less lattice sites. The actual position of the crossover depends on computational cost of the overhead relative to the number of lattice sites being sampled. Comparing the three graphs in the figure, we note that the crossover shear

[9] By an nth-neighbor model, we mean a choice of cutoff radius that in the reference configuration contains all lattice sites up to and including the nth nearest-neighbor shell of lattice sites to the origin.

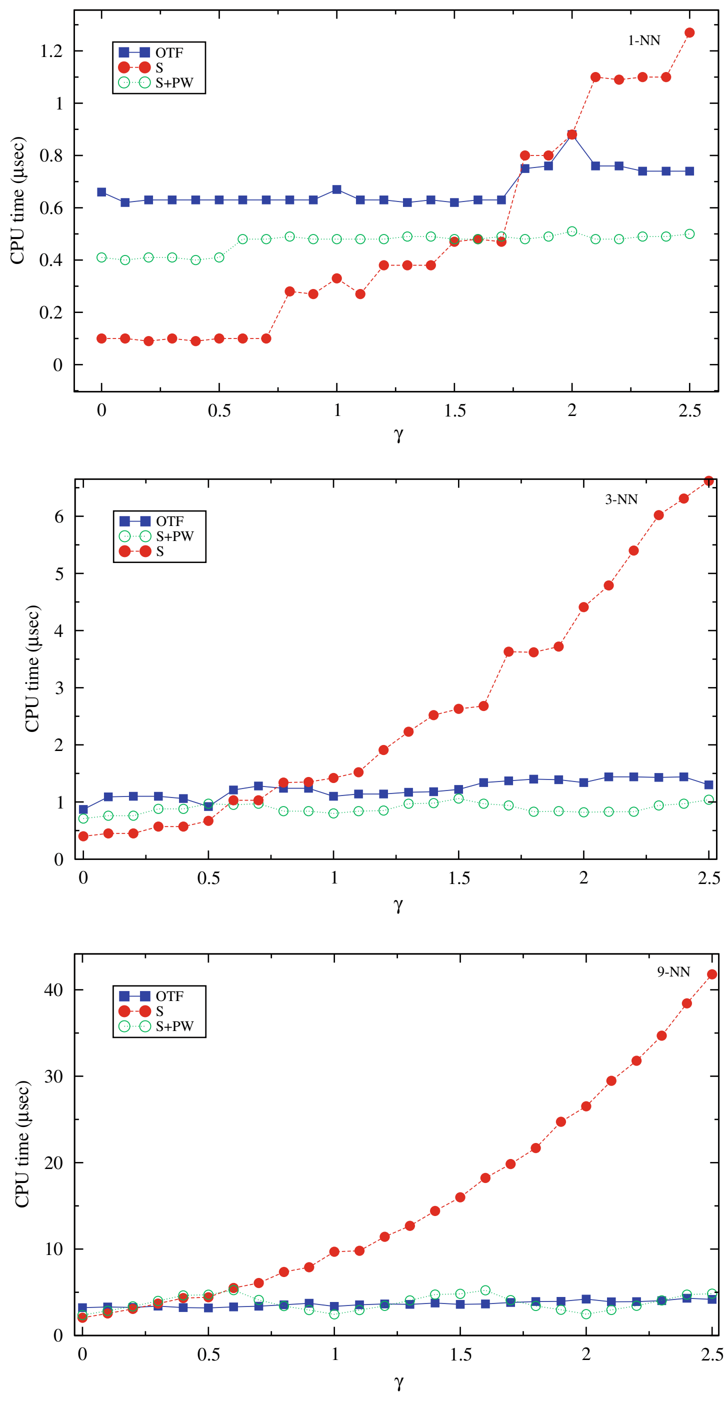


Fig. 4. Cluster construction in simple lattices undergoing simple shear. The time required to find all lattice sites within a given cutoff radius of the origin as a function of the applied shear γ. Three different radii are considered corresponding to a nearest-neighbor mode (1-NN), third-neighbor model (3-NN), and ninth-neighbor model (9-NN).

reduces with increasing model size. For a 1-NN model, the crossover shear is $\gamma \simeq 1.5$, for a 3-NN model it is $\gamma \simeq 0.6$, and for a 9-NN model it is $\gamma \simeq 0.2$. The fact that the crossover shear is reducing with increasing model size is a consequence of the slightly better scaling properties of OTF and S + PW relative to S. The conclusion is that the choice of the optimal method depends on the model size and the typical shears that are expected in the application. However, since for a physical system the crossover shears measured above would be considered large, method S may be the optimal choice for simple lattices in practical applications.

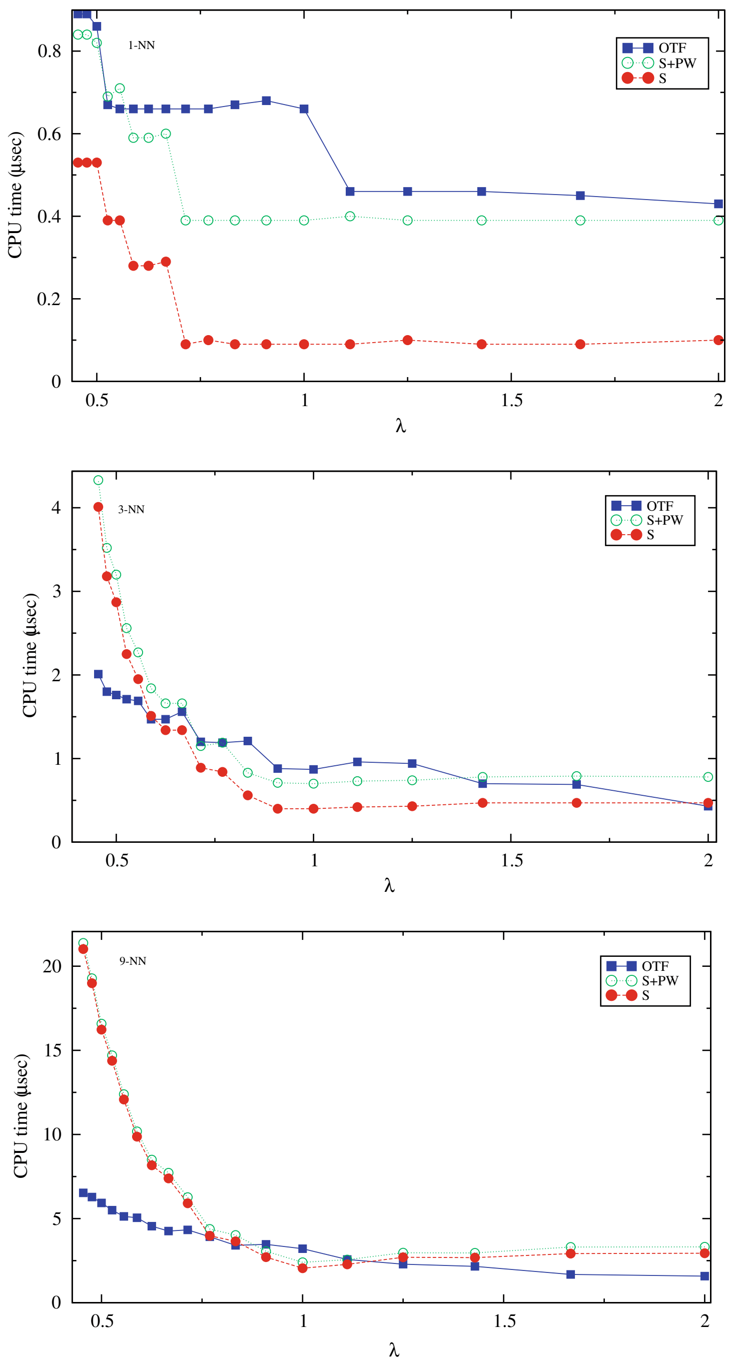


Fig. 5. Cluster construction in simple lattices undergoing uniaxial stretch. The time required to find all lattice sites within a given cutoff radius of the origin as a function of the applied stretch λ. Three different radii are considered corresponding to a nearest-neighbor mode (1-NN), third-neighbor model (3-NN), and ninth-neighbor model (9-NN).

Next, we turn to uniaxial stretch deformations. We explored stretches in the range $\lambda \in [0.4, 2.0]$. The results of the timing study for uniaxial stretch are shown in Fig. 5. As for simple shear, three different models were tested on three different model sizes. The total computation time as a function of λ is plotted. We note that all method are insensitive to stretch for $\lambda > 1$ but require more time as λ decreases below one. The reason is that for $\lambda < 1$ the lattice is compressed and more lattice sites are entering the cutoff sphere, whereas for $\lambda > 1$ the lattice is stretched and the number of lattice sites in the cutoff sphere is reduced. The entrance of new lattice sites into the cutoff sphere is particularly clear in the 1-NN model where such events are accompanied by large discontinuous jumps in the computation time. For larger model sizes the effect is smeared out by the more continuous entry of lattice sites into the cutoff sphere with compression.

Table 2
Cluster construction in randomly deformed simple lattices. Running times for the individual steps involved in finding all lattice sites within the cutoff sphere for a 5-NN model. Times are averages over 100 randomly-generated deformation gradients with $R_i \leq 20$. Computations were performed on an Intel Core2 Q6600 2.40 GHz CPU.

Method	Setup (μs)	Reduction (μs)	Construction (μs)
OTF	N/A	N/A	2.06
S	2718.92	N/A	30.74
S + PW	2718.92	0.48	2.30

Similar to the case of simple shear, no single method is best for all model sizes and stretch parameters λ. In this case, lattice reduction does not save CPU time, since for the uniaxial stretch deformation the distorted lattice $\boldsymbol{FA}$ is already reduced. For this reason the S + PW curves are always above the S curves with the difference equal to the time required for the PW algorithm. The performance of the remaining two methods, S and OTF, depends on the model size. For the nearest-neighbor model (1-NN), the shell method S is most efficient. However, as the model size increases OTF becomes more competitive. For the third-neighbor model (3-NN), method S still remains the optimal choice except under very large compression ($\lambda < 0.6$) when OTF outperforms it. For the ninth-neighbor model (9-NN), OTF outperforms S over most of the deformation range except near the undeformed state ($0.8 < \lambda < 1.2$).

The timing studies described so far consider idealized cases involving shear or stretch deformations separately. As a final test for simple lattices, we consider the more general case of randomly-generated deformation gradients. One hundred random matrices with an influence radius $R_i \leqslant 20$ are generated to sample different possible combinations of shear and stretch deformations. The cutoff radius is set to include five nearest-neighbor shells (5-NN). The results are tabulated in Table 2 where the time necessary to setup the shells and the average times for lattice reduction and for cluster construction are shown. As can be inferred from the table, the setup stage of the shell methods takes an appreciably large amount of time compared to the time necessary to construct the cluster, while the lattice reduction time is relatively insignificant. Of course, the setup stage only needs to be performed once for a given lattice and is therefore normally inconsequential. Note that the efficiency of OTF and S+PW is comparable with only a slight advantage for OTF, while method S is not competitive.

4.3. Timing results for multilattices

In this section, we discuss the efficiency of methods S, S + PW, and OTF when applied to multilattices. The nominal lattice is the same cubic lattice used in the previous section. We apply the same shear and stretch deformations to the multilattice as described in the preceding section for simple lattices and examine the performance of the three methods for various model sizes. Our calculations indicate that the methods are insensitive to the number of sublattices. We therefore only present results for multilattices with two sublattices (2-lattices). To make our timing studies independent of the particular choice of the sublattice positions, the latter are chosen at random, and the timing results reflect the average over a set of many multilattices obtained in this manner. We determined that a set of 600 randomly distributed samples were sufficient to obtain a converged average. The scatter about the average is displayed in the following graphs as error bars. To accurately measure the CPU time every cluster construction is repeated 10^4 times.

The timing results for the simple shear deformation are displayed in Fig. 6. Note that in contrast to simple lattices, OTF is the most efficient method for any model size and any shear parameter γ. The second best method is S + PW. The additional computational cost for lattice reduction pays off by considerably decreasing the CPU time relative to method S, especially for large values of γ. As before, this is due to lattice-invariant shears. The zigzag form of the S + PW curve resulting from this is particularly clear here.

In addition to being the fastest method, OTF also has the advantage that the scatter due to sublattice positions is minimal. In contrast, both S and S + PW exhibit large scatter. This is due to a noticeable variation in the number of shells that are taken into account for large γ. The number of shells and the influence radius R_i, see (39), depends on the sublattice positions, and therefore varies appreciably when the sublattice positions are sampled at random.

Fig. 7 show the results for uniaxial stretch. The timing results S + PW are not displayed since lattice reduction does not reduce CPU time just as in the case of simple lattices. The conclusions for this deformation are the same as for simple shear. OTF is the most efficient method for cluster construction for any model size and any stretch parameter λ. The scatter due to sublattice positions remains minimal for OTF, whereas it is significant for method S.

Finally, we consider the general case of randomly-generated deformation gradients. One hundred random matrices with an influence radius $R_i(\boldsymbol{F}, \boldsymbol{\delta}_{\max}, r) \leqslant 20$ are generated to sample different possible combinations of shear and stretch deformations. The cutoff radius is set to include five nearest-neighbor shells (5-NN). The results are tabulated in Table 3 where the time necessary to setup the shells and the average times for lattice reduction and for cluster construction are shown. The conclusions regarding the setup and reduction times are the same as for simple lattices described earlier. The main conclusions are that OTF outperforms S + PW by about a factor of two on average and that method S without lattice reduction is not competitive.[10]

[10] As noted earlier, tests performed on multilattices with larger numbers of sublattices do not change our conclusions. OTF remains the most efficient method for multilattices regardless of the number sublattices.

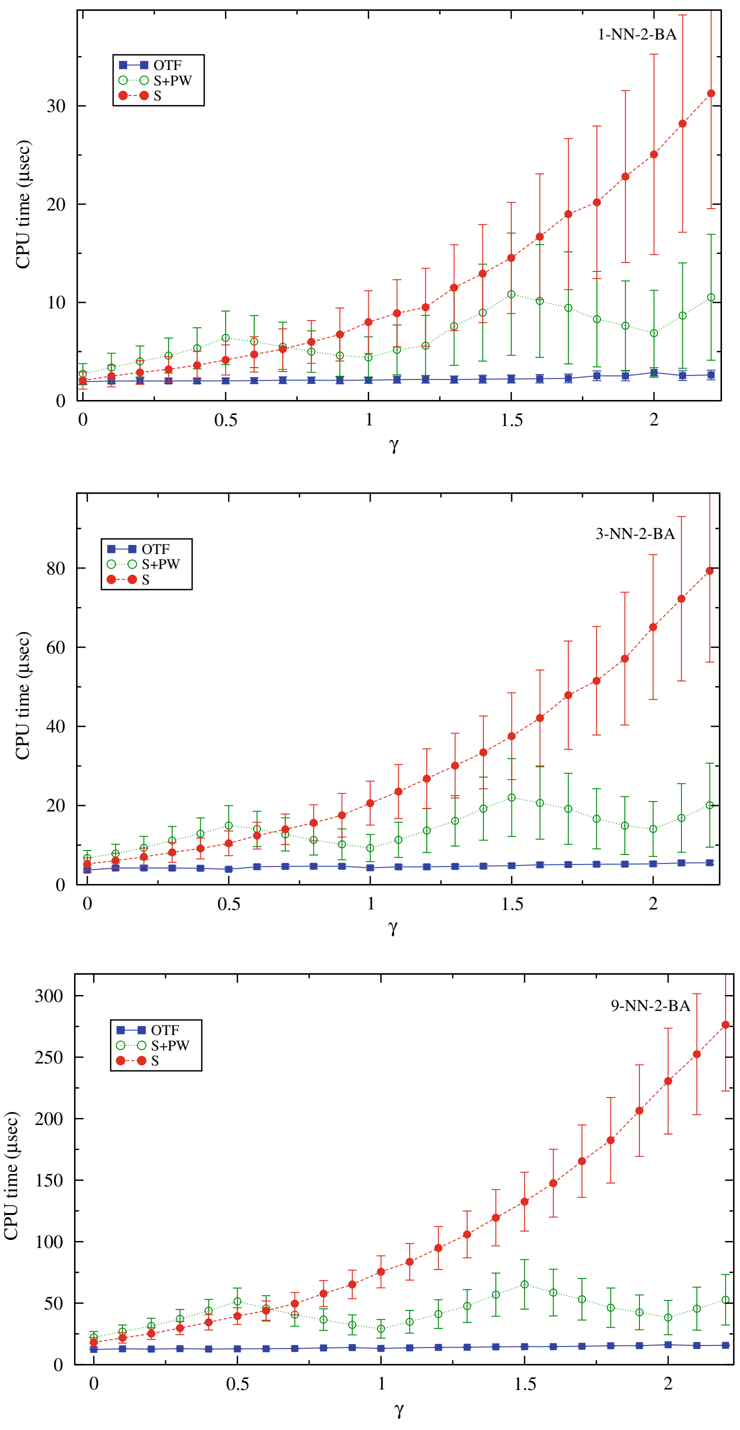


Fig. 6. Cluster construction in 2-lattices undergoing simple shear. The time required to find all lattice sites within a given cutoff radius of the origin as a function of the applied shear γ. Three different radii are considered corresponding to a nearest-neighbor mode (1-NN), third-neighbor model (3-NN), and ninth-neighbor model (9-NN). The bars reflect the range of times measured for 600 randomly-generated sublattice positions.

5. Conclusions

We have presented a set of efficient algorithms for performing several tasks related to computations involving simple lattices and multilattices, namely lattice reduction, detection of nearest lattice sites, and computation of clusters of nearest neighbors. The performance of the algorithms was numerically studied for prototypical test cases that are relevant for the simulation of crystalline solids.

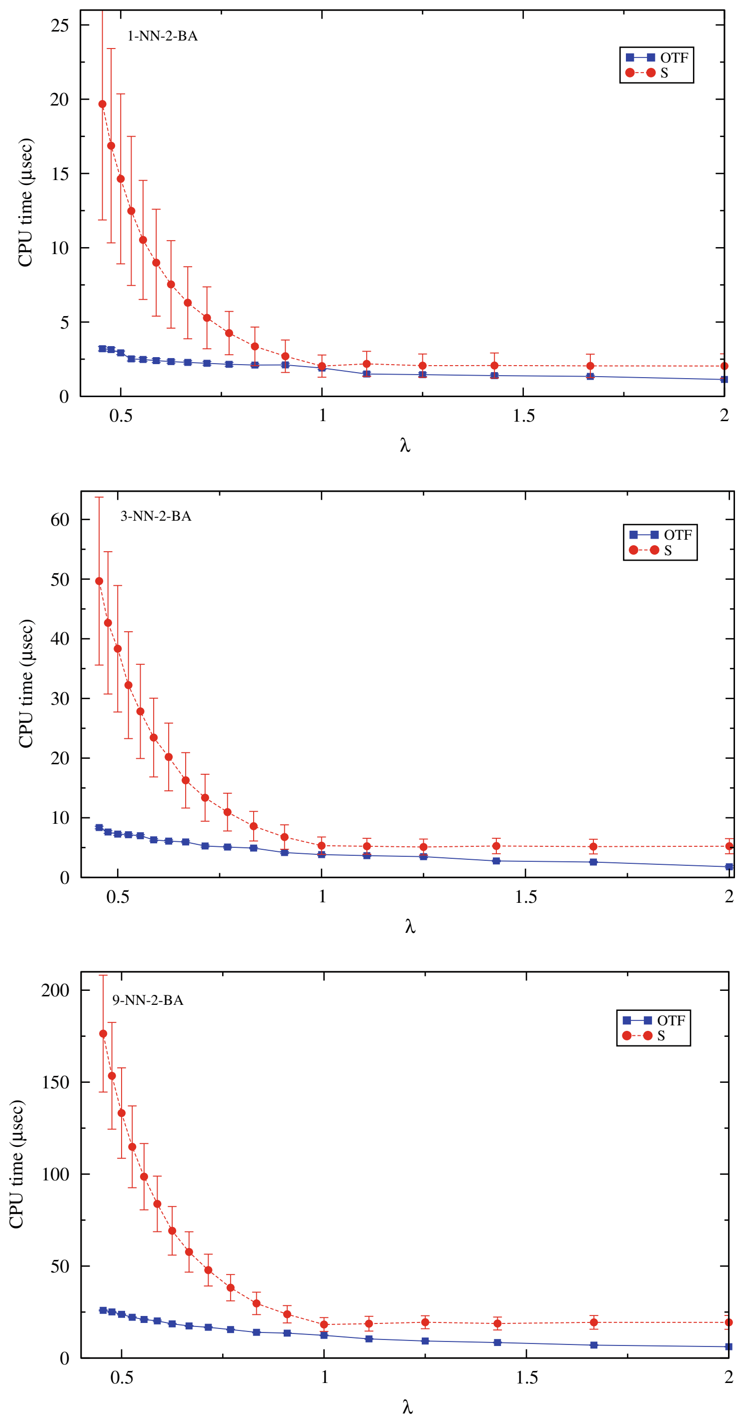


Fig. 7. Cluster construction in 2-lattices undergoing uniform stretch. The time required to find all lattice sites within a given cutoff radius of the origin as a function of the applied stretch λ. Three different radii are considered corresponding to a nearest-neighbor mode (1-NN), third-neighbor model (3-NN), and ninth-neighbor model (9-NN). The bars reflect the range of times measured for 600 randomly-generated sublattice positions.

We discussed two algorithms for lattice reduction: LLL reduction and pairwise reduction. Both compute an optimized set of lattice vectors that are used to speed up subsequent algorithms. The LLL algorithm leads to an approximate global optimum in terms of the orthogonality of the lattice vectors, whereas the pairwise algorithm results in a set of lattice vectors that are pairwise shortest and most orthogonal. The pairwise algorithm is far easier to implement than LLL and performs nearly as well. In addition, the clear physical significance of the resulting pairwise reduced lattice vectors is an advantage.

Table 3
Cluster construction in randomly deformed 2-lattices. Running times for the individual steps involved in finding all lattice sites within the cutoff sphere for a 5-NN model. Times are averages over 100 randomly-generated deformation gradients with $R_i(\boldsymbol{F}, \boldsymbol{\delta}_{\max}, r) \leqslant 20$. Statistics is collected over 600 samples with randomly positioned sublattices. Computations were performed on an Intel Core2 Q6600 2.40 GHz CPU.

Method	Setup (μs)	Reduction (μs)	Construction (μs)
OTF	N/A	N/A	17.33 ± 0.8
S	2719.34	N/A	147.53 ± 16.9
S + PW	2719.34	6.3 ± 0.1	39.24 ± 5.8

The next problem studied was the detection of the nearest lattice site to a specified arbitrary point in space. We explored two approaches: a naive brute-force algorithm and a new short-list algorithm. The brute-force algorithm is based on scanning over a (possibly large) set of lattice sites that is guaranteed to contain the desired site. In contrast, the short-list algorithm dramatically reduces the set of candidate lattice sites by performing a preprocessing step. The preprocessing step is computationally expensive, however it needs to performed only once and it greatly reduces the overall computation time in situations where many detection operations on the same lattice are performed.

The final problem studied was the computation of nearest-neighbor clusters. This involves the detection of all lattice sites within a given radius of specified lattice point for both simple lattices and multilattices. Two methods were developed and explored: the shell algorithm and the on-the-fly algorithm. The shell algorithm is based on checking a set of candidates from a pre-computed sorted crystallite, whereas the on-the-fly algorithm provides loops that directly provide the required set of lattice sites at the cost of computing complex loop bounds. The performance of the shell method is greatly improved if the lattice is reduced in advance. For simple lattices that are not too deformed the shell method is favorable, whereas for more heavily deformed simple lattices and for multilattices, the on-the-fly method outperforms it by a factor of two on average.

We conclude that the choice of the optimal algorithm for a certain lattice task highly depends on the setting and the problem data. Even an apparently simple approach, such as the brute-force algorithm for nearest lattice site detection, can be superior to more advanced methods in certain situations. It is therefore essential to estimate the expected problem data for the respective application in advance and then to choose the appropriate algorithm.

The tasks we dealt with in this paper only represent a small subset of tasks that are frequently encountered during lattice computations. One example is the determination of the essential unit cell for a multilattice, *i.e.*, given an arbitrary multilattice description, to find an optimized description of the multilattice where the number of fractional position vectors is smallest and their lengths are shortest. Another common task is given a triangulation where the nodes are lattice sites, to find the cell that contains an arbitrary specified lattice site. An efficient algorithm would exploit the underlying lattice structure to speed up the search compared to classical approaches like oct-tree methods for triangulations with arbitrary nodes. However, these tasks and others are beyond the scope of this paper.

Acknowledgments

This work was supported in part by the Department of Energy under Award Number DE-FG02-05ER25706 and by the University of Minnesota Supercomputing Institute. The authors acknowledge preliminary work performed by Jon Wilkening on the "short-list" method, while working as a postdoctoral fellow on quasicontinuum applications. The authors would like to thank Ryan Elliott for his careful reading of the manuscript and many helpful comments.

Appendix A

Several proofs have been omitted in the text in order not to interrupt the flow by too many technical details. We make up for this now.

A.1. Pairwise lattice reduction

First we start with two properties from Section 2 regarding length reduction and angle reduction.

Lemma A.1. *Let $\boldsymbol{a}_i, \boldsymbol{a}_j \in \mathbb{R}^d$ be two linearly independent vectors. If $\boldsymbol{a}_i$ is length-reduced by $\boldsymbol{a}_j$, that is*

$$\|\boldsymbol{a}_i - m\boldsymbol{a}_j\| \geqslant \|\boldsymbol{a}_i\| \tag{A.1}$$

for all $m \in \mathbb{Z}$, then $\boldsymbol{a}_i$ is angle-reduced to $\boldsymbol{a}_j$ as well, i.e., $\boldsymbol{a}_i$ is the vector among all $\boldsymbol{a}_i - m\boldsymbol{a}_j$ whose angle with $\boldsymbol{a}_j$ is closest to 90° or $\frac{\pi}{2}$ in radians,

$$\left|\angle(\boldsymbol{a}_i - m\boldsymbol{a}_j, \boldsymbol{a}_j) - \frac{\pi}{2}\right| \geqslant \left|\angle(\boldsymbol{a}_i, \boldsymbol{a}_j) - \frac{\pi}{2}\right| \tag{A.2}$$

for all $m \in \mathbb{Z}$.

Proof. Because

$$\cos\angle(\boldsymbol{a}_i - m\boldsymbol{a}_j, \boldsymbol{a}_j) = \frac{|(\boldsymbol{a}_i - m\boldsymbol{a}_j)\cdot\boldsymbol{a}_j|}{\|\boldsymbol{a}_i - m\boldsymbol{a}_j\|\|\boldsymbol{a}_j\|}, \tag{A.3}$$

we need to show that (A.1) is equivalent to

$$\frac{|(\boldsymbol{a}_i - m\boldsymbol{a}_j)\cdot\boldsymbol{a}_j|}{\|\boldsymbol{a}_i - m\boldsymbol{a}_j\|\|\boldsymbol{a}_j\|} \geqslant \frac{|\boldsymbol{a}_i\cdot\boldsymbol{a}_j|}{\|\boldsymbol{a}_i\|\|\boldsymbol{a}_j\|} \tag{A.4}$$

for all $m \in \mathbb{Z}$. Squaring and multiplying out the terms gives that (A.4) is equivalent to

$$\left((\boldsymbol{a}_i\cdot\boldsymbol{a}_j)^2 - 2m(\boldsymbol{a}_i\cdot\boldsymbol{a}_j)\|\boldsymbol{a}_j\|^2 + m^2\|\boldsymbol{a}_j\|^4\right)\|\boldsymbol{a}_i\|^2 \geqslant (\boldsymbol{a}_i\cdot\boldsymbol{a}_j)^2\left(\|\boldsymbol{a}_i\|^2 - 2m(\boldsymbol{a}_i\cdot\boldsymbol{a}_j) + m^2\|\boldsymbol{a}_j\|^2\right). \tag{A.5}$$

Rearranging terms leads to

$$\left(-2m(\boldsymbol{a}_i\cdot\boldsymbol{a}_j) + m^2\|\boldsymbol{a}_j\|^2\right)\left(\|\boldsymbol{a}_i\|^2\|\boldsymbol{a}_j\|^2 - (\boldsymbol{a}_i\cdot\boldsymbol{a}_j)^2\right) \geqslant 0. \tag{A.6}$$

The second factor on the left side of the inequality is always positive because $\boldsymbol{a}_i$ and $\boldsymbol{a}_j$ are linearly independent. Therefore (A.6) is equivalent to

$$-2m(\boldsymbol{a}_i\cdot\boldsymbol{a}_j) + m^2\|\boldsymbol{a}_j\|^2 \geqslant 0. \tag{A.7}$$

Adding $\|\boldsymbol{a}_i\|^2$ to both sides shows that (A.7) is equivalent to (A.1). □

Lemma A.2. *If some longer vector $\boldsymbol{a}_l \in \mathbb{R}^d$ is length-reduced by some shorter vector $\boldsymbol{a}_s \in \mathbb{R}^d$, i.e., $\|\boldsymbol{a}_l\| \geq \|\boldsymbol{a}_s\|$ and $\|\boldsymbol{a}_l - m\boldsymbol{a}_s\| \geq \|\boldsymbol{a}_l\|$ for all $m \in \mathbb{Z}$, then the shorter vector $\boldsymbol{a}_s$ is automatically length-reduced by the longer vector $\boldsymbol{a}_l$ as well, i.e., $\|\boldsymbol{a}_s - m\boldsymbol{a}_l\| \geq \|\boldsymbol{a}_s\|$ for all $m \in \mathbb{Z}$.*

Proof. We have that

$$\|\boldsymbol{a}_s - m\boldsymbol{a}_l\|^2 - \|\boldsymbol{a}_s\|^2 = -2m(\boldsymbol{a}_s\cdot\boldsymbol{a}_l) + m^2\|\boldsymbol{a}_l\|^2 \geqslant -2m(\boldsymbol{a}_s\cdot\boldsymbol{a}_l) + m^2\|\boldsymbol{a}_s\|^2 = \|\boldsymbol{a}_l - m\boldsymbol{a}_s\|^2 - \|\boldsymbol{a}_l\|^2 \geqslant 0. \quad \square \tag{A.8}$$

A.2. Search range for short-list method

Next, we come to the derivation of the search range, $\mathcal{R}_{sl}$, for the short-list method described in Section 3.2. Recall that $\mathcal{R}_{sl}$ consists of (at least) all lattice sites that are closest to at least one point in $\boldsymbol{A}\left[-\frac{1}{2},\frac{1}{2}\right]^d$. The search range, $\mathcal{R}_{sl}$, needs to be determined in a preprocessing step.

The key to this preprocessing step are Voronoi cells, also referred to as Wigner–Seitz cells in physics. The Voronoi cell $\mathcal{V}(\boldsymbol{n})$ of some lattice point $\boldsymbol{An}$ is defined as

$$\mathcal{V}(\boldsymbol{n}) := \left\{\boldsymbol{x} \in \mathbb{R}^d : \|\boldsymbol{An} - \boldsymbol{x}\| \leqslant \|\boldsymbol{Am} - \boldsymbol{x}\| \ \forall \boldsymbol{m} \in \mathbb{Z}^d\right\}, \tag{A.9}$$

i.e., the set of all points $\boldsymbol{x}$ to which $\boldsymbol{An}$ is the closest lattice point. It can be shown that $\mathcal{V}(\boldsymbol{n})$ is the intersection

$$\mathcal{V}(\boldsymbol{n}) = \bigcap_{\boldsymbol{m}\in\mathbb{Z}^d, \boldsymbol{m}\neq\boldsymbol{n}} \mathcal{H}(\boldsymbol{n},\boldsymbol{m}) \tag{A.10}$$

of half-spaces $\mathcal{H}(\boldsymbol{n},\boldsymbol{m})$ defined by (24). Note that the half-space $\mathcal{H}(\boldsymbol{n},\boldsymbol{m})$ contains $\boldsymbol{An}$ and is bounded by the plane that is perpendicular to the line from $\boldsymbol{An}$ to $\boldsymbol{Am}$ and that goes through the midpoint of $\boldsymbol{An}$ and $\boldsymbol{Am}$. The construction of the Voronoi cell from half-spaces is illustrated in Fig. A.1. For a detailed characterization and classification of Voronoi cells and a Voronoi reduction algorithm, see [3].

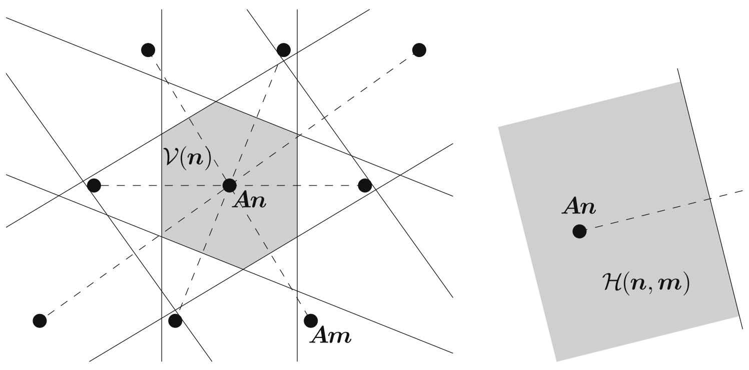
Using Voronoi cells, the optimal search range can be characterized as the set of all $\boldsymbol{n} \in \mathbb{Z}^d$ for which at least one point $\bar{\boldsymbol{x}} \in \boldsymbol{A}\left[-\frac{1}{2},\frac{1}{2}\right]^d$ is contained in $\mathcal{V}(\boldsymbol{n})$, *i.e.*, $\bar{\boldsymbol{x}} \in \mathcal{V}(\boldsymbol{n})$. In other words, the optimal search range is the set $\mathcal{R}_{opt}$ of lattice points whose Voronoi cells intersect with the region $\boldsymbol{A}[-\frac{1}{2},\frac{1}{2}]^d$,

$$\mathcal{R}_{opt} = \left\{\boldsymbol{m} \in \mathbb{Z}^d : \boldsymbol{A}[-\tfrac{1}{2},\tfrac{1}{2}]^d \cap \mathcal{V}(\boldsymbol{m}) \neq \emptyset\right\}. \tag{A.11}$$

A calculation similar to (17) shows that all $\boldsymbol{m} \in \mathcal{R}_{opt}$ satisfy $\|\boldsymbol{m}\|_\infty \leqslant R_b(\boldsymbol{A})$, or equivalently, $\boldsymbol{m} \in \mathcal{R}_{bf}$, where $\mathcal{R}_{bf}$ is defined in (22). Thus,

$$\mathcal{R}_{opt} = \left\{\boldsymbol{m} \in \mathcal{R}_{bf} : \boldsymbol{A}[-\tfrac{1}{2},\tfrac{1}{2}]^d \cap \mathcal{V}(\boldsymbol{m}) \neq \emptyset\right\}, \tag{A.12}$$

which limits the set of possible $\boldsymbol{m}$ to a finite number.



$\mathcal{V}(\boldsymbol{n})$
$\boldsymbol{An}$
$\boldsymbol{Am}$
$\boldsymbol{Am}$
$\boldsymbol{An}$
$\mathcal{H}(\boldsymbol{n},\boldsymbol{m})$

Fig. A.1. Voronoi cell, $\mathcal{V}(\boldsymbol{n})$, and its construction from half-spaces, $\mathcal{H}(\boldsymbol{n},\boldsymbol{m})$.

Since Voronoi cells are expensive to compute, we determine a superset $\mathcal{R}_{sl}$ of $\mathcal{R}_{opt}$ that is faster to compute but not much larger in practice. From the description (A.10) of the Voronoi cell from half-spaces, we have

$$\mathcal{R}_{opt} = \left\{\boldsymbol{m} \in \mathcal{R}_{bf} : \exists \boldsymbol{x} \in \boldsymbol{A}[-\tfrac{1}{2},\tfrac{1}{2}]^d : \boldsymbol{x} \in \mathcal{V}(\boldsymbol{m})\right\} = \left\{\boldsymbol{m} \in \mathcal{R}_{bf} : \exists \boldsymbol{x} \in \boldsymbol{A}[-\tfrac{1}{2},\tfrac{1}{2}]^d \forall \boldsymbol{m}' \in \mathbb{Z}^d, \boldsymbol{m}' \neq \boldsymbol{m} : \boldsymbol{x} \in \mathcal{H}(\boldsymbol{m},\boldsymbol{m}')\right\}. \tag{A.13}$$

Just as for $\boldsymbol{m}$, we also restrict the set of indices $\boldsymbol{m}' \in \mathbb{Z}^d$ in the condition above to a finite set, $\boldsymbol{m} \in \mathcal{R}_{bf}$. This way, the condition above gets less strict, and we enlarge the set $\mathcal{R}_{opt}$,

$$\mathcal{R}_{opt} \subset \left\{\boldsymbol{m} \in \mathcal{R}_{bf} : \exists \boldsymbol{x} \in \boldsymbol{A}[-\tfrac{1}{2},\tfrac{1}{2}]^d \forall \boldsymbol{m}' \in \mathcal{R}_{bf}, \boldsymbol{m}' \neq \boldsymbol{m} : \boldsymbol{x} \in \mathcal{H}(\boldsymbol{m},\boldsymbol{m}')\right\}. \tag{A.14}$$

The crucial step is to relax the condition that a single point, $\boldsymbol{x}$, is contained in all half-spaces $\mathcal{H}(\boldsymbol{m},\boldsymbol{m}')$. By exchanging the order of the existential quantifier, $\exists$, and the universal quantifier, $\forall$, we allow for a different point $\boldsymbol{x}$ for each lattice point $\boldsymbol{m}'$. This way, we get the estimate $\mathcal{R}_{opt} \subset \mathcal{R}_{sl}$ where

$$\begin{aligned}\mathcal{R}_{opt} &\subset \left\{\boldsymbol{m} \in \mathcal{R}_{bf} : \exists \boldsymbol{x} \in \boldsymbol{A}[-\tfrac{1}{2},\tfrac{1}{2}]^d \forall \boldsymbol{m}' \in \mathcal{R}_{bf}, \boldsymbol{m}' \neq \boldsymbol{m} : \boldsymbol{x} \in \mathcal{H}(\boldsymbol{m},\boldsymbol{m}')\right\} \\ &\subset \left\{\boldsymbol{m} \in \mathcal{R}_{bf} : \forall \boldsymbol{m}' \in \mathcal{R}_{bf}, \boldsymbol{m}' \neq \boldsymbol{m}\ \exists \boldsymbol{x} \in \boldsymbol{A}[-\tfrac{1}{2},\tfrac{1}{2}]^d : \boldsymbol{x} \in \mathcal{H}(\boldsymbol{m},\boldsymbol{m}')\right\} =: \mathcal{R}_{sl}.\end{aligned} \tag{A.15}$$

Whenever the half-space $\mathcal{H}(\boldsymbol{m},\boldsymbol{m}')$ contains a point $\boldsymbol{x} \in \boldsymbol{A}[-\frac{1}{2},\frac{1}{2}]^d$, it always contains at least one corner of $\boldsymbol{A}[-\frac{1}{2},\frac{1}{2}]^d$, *i.e.*, one element of $\boldsymbol{A}\{-\frac{1}{2},\frac{1}{2}\}^d$. Thus, we have

$$\begin{aligned}\mathcal{R}_{sl} &= \left\{\boldsymbol{m} \in \mathcal{R}_{bf} : \forall \boldsymbol{m}' \in \mathcal{R}_{bf}, \boldsymbol{m}' \neq \boldsymbol{m}\ \exists \boldsymbol{x} \in \boldsymbol{A}\{-\tfrac{1}{2},\tfrac{1}{2}\}^d : \boldsymbol{x} \in \mathcal{H}(\boldsymbol{m},\boldsymbol{m}')\right\} \\ &= \left\{\boldsymbol{m} \in \mathcal{R}_{bf} : \forall \boldsymbol{m}' \in \mathcal{R}_{bf}, \boldsymbol{m}' \neq \boldsymbol{m} : \boldsymbol{A}\{-\tfrac{1}{2},\tfrac{1}{2}\}^d \cap \mathcal{H}(\boldsymbol{m},\boldsymbol{m}') \neq \emptyset\right\},\end{aligned} \tag{A.16}$$

which is precisely the definition (23) of $\mathcal{R}_{sl}$ in Section 3.2. Note that (A.16) is based on a finite number of possible vectors $\boldsymbol{m}, \boldsymbol{m}'$ and corner points $\boldsymbol{x}$ that need to be checked against each other by the half-space condition, $\boldsymbol{x} \in \mathcal{H}(\boldsymbol{m},\boldsymbol{m}')$. This makes the search range actually computable. See Algorithm 3.

A.3. On-the-fly method for cluster computation

Finally, we come to a property used in the on-the-fly algorithm for cluster computation in Section 4.1.2.

Lemma A.3. *Let*

$$\boldsymbol{G} = \begin{bmatrix} \tilde{\boldsymbol{G}} & \boldsymbol{g} \\ \boldsymbol{g}^T & \gamma \end{bmatrix} \in \mathbb{R}^{d\times d} \tag{A.17}$$

with $\tilde{\boldsymbol{G}} \in \mathbb{R}^{(d-1)\times(d-1)}, \boldsymbol{g} \in \mathbb{R}^{d-1}, \gamma \in \mathbb{R}$ *be a symmetric positive definite matrix. Then*

$$\hat{\boldsymbol{G}} := \tilde{\boldsymbol{G}} - \frac{\boldsymbol{g} \otimes \boldsymbol{g}}{\gamma} \in \mathbb{R}^{(d-1)\times(d-1)} \tag{A.18}$$

is symmetric positive definite as well.

Proof. $\hat{\boldsymbol{G}}$ is symmetric by definition. We need to show that $\tilde{\boldsymbol{x}}^T\hat{\boldsymbol{G}}\tilde{\boldsymbol{x}} > 0$ for any given non-zero vector $\tilde{\boldsymbol{x}} \in \mathbb{R}^{d-1}$. Define

$$\boldsymbol{x} := \begin{bmatrix} \tilde{\boldsymbol{x}} \\ \xi \end{bmatrix} \in \mathbb{R}^d \quad \text{with } \xi := -\frac{\boldsymbol{g}^T\tilde{\boldsymbol{x}}}{\gamma} \in \mathbb{R}. \tag{A.19}$$

Then $\boldsymbol{x}\neq 0$, and $\boldsymbol{x}^T G\boldsymbol{x} > 0$ because G is positive definite. We thus have

$$\tilde{\boldsymbol{x}}^T\hat{\boldsymbol{G}}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{x}}^T\tilde{\boldsymbol{G}}\tilde{\boldsymbol{x}} - \frac{(\mathbf{g}^T\tilde{\boldsymbol{x}})^2}{\gamma} = \tilde{\boldsymbol{x}}^T\tilde{\boldsymbol{G}}\tilde{\boldsymbol{x}} + 2\xi\mathbf{g}^T\tilde{\boldsymbol{x}} + \gamma\xi^2 = \boldsymbol{x}^T\boldsymbol{G}\boldsymbol{x} > 0, \tag{A.20}$$

thus $\hat{\boldsymbol{G}}$ is positive definite. □

References

[1] E. Agrell, T. Eriksson, A. Vardy, K. Zeger, Closest point search in lattices, IEEE Trans. Inf. Theory 48 (8) (2002) 2201–2214.
[2] J.W.S. Cassels, An Introduction to the Geometry of Numbers, Grundlehren der mathematischen Wissenschaften in Einzeldarstellungen mit besonderer Berücksichtigung der Anwendungsgebiete, vol. 99, Springer-Verlag, Berlin/Heidelberg/New York, 1971.
[3] J.H. Conway, N.J.A. Sloane, Low-dimensional lattices. VI. Voronoi reduction of three-dimensional lattices, Proc. Roy. Soc. Lond. A 436 (1896) (1992) 55–68.
[4] M. Dobson, R.S. Elliott, M. Luskin, E.B. Tadmor, A multilattice quasicontinuum for phase transforming materials: Cascading Cauchy Born kinematics, J. Comput. Aided Mater. Des. 14 (Suppl. 1) (2007) 219–237.
[5] C.F. Gauss, Disquisitiones Arithmeticae (A.A. Clarke, Trans., 1966), Yale University Press, New Haven, London, 1801.
[6] R. Kannan, Algorithmic geometry of numbers, Annu. Rev. Comput. Sci. 2 (1987) 231–267.
[7] K. Kremer, K. Binder, Monte Carlo simulations of lattice models for macromolecules, Comp. Phys. Rep. 7 (1988) 259–310.
[8] A.K. Lenstra, H.W. Lenstra Jr., L. Lovász, Factoring polynomials with rational coefficients, Math. Ann. 261 (1982) 515–534.
[9] B.M. McCoy, T.T. Wu, The Two-Dimensional Ising Model, Harvard University Press, Cambridge, 1973.
[10] G.P. Parry, On essential and non-essential descriptions of multilattices, Math. Mech. Solids 9 (2004) 411–418.
[11] M. Pitteri, Geometry and symmetry of multilattices, Int. J. Plast. 14 (1998) 139–157.
[12] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes in FORTRAN. The Art of Scientific Computing, second ed., Cambridge University Press, Cambridge, 1992.
[13] V.B. Shenoy, R. Miller, E.B. Tadmor, D. Rodney, R. Phillips, M. Ortiz, An adaptive finite element approach to atomic-scale mechanics—the quasicontinuum method, J. Mech. Phys. Solids 47 (3) (1999) 611–642.
[14] E.B. Tadmor, M. Ortiz, R. Phillips, Quasicontinuum analysis of defects in solids, Philos. Mag. A 73 (6) (1996) 1529–1563.
[15] E.B. Tadmor, G.S. Smith, N. Bernstein, E. Kaxiras, Mixed finite element and atomistic formulation for complex crystals, Phys. Rev. B 59 (1) (1999) 235–245.
[16] K. Yue, K.M. Fiebig, P.D. Thomas, H.S. Chan, E.I. Shakhnovich, K.A. Dill, A test of lattice protein folding algorithms, Proc. Natl. Acad. Sci. USA 92 (1995) 325–329.