SIAM J. DISCRETE MATH.
Vol. 23, No. 2, pp. 715–731

FINDING THE CLOSEST LATTICE POINT BY ITERATIVE SLICING*

NAFTALI SOMMER†, MEIR FEDER†, AND OFIR SHALVI†

Abstract. Most of the existing methods that are used to solve the closest lattice point problem are based on an efficient search of the lattice points. In this paper a novel alternative approach is suggested where the closest point to a given vector is found by calculating which Voronoi cell contains this vector in an iterative manner. Each iteration is made of simple "slicing" operations, using a list of the Voronoi relevant vectors that define the basic Voronoi cell of the lattice. The algorithm is guaranteed to converge to the closest lattice point in a finite number of steps. The method is suitable, for example, for decoding of multi-input multi-output (MIMO) communication problems. The average computational complexity of the proposed method is comparable to that of the efficient variants of the sphere decoder, but its computational variability is smaller.

Key words. lattice, closest neighbor, Voronoi relevant vectors, closest point search, lattice decoding, lattice quantization, lattice codes

AMS subject classifications. 03G10, 94B60, 94A05

DOI. 10.1137/060676362

1. Introduction.

A real lattice of dimension n is defined as the set of all linear combinations of n real basis vectors, where the coefficients of the linear combination are restricted to be integers:

$$\Lambda(\boldsymbol{G}) \triangleq \{\boldsymbol{G}\underline{u} : \underline{u} \in \mathbb{Z}^n\}.$$

$\boldsymbol{G}$ is the $m \times n$ generator matrix of the lattice, whose columns are the basis vectors which are assumed to be linearly independent over $\mathbb{R}^m$. The closest lattice point problem is the problem of finding, for a given lattice Λ and a given input point $\underline{x} \in \mathbb{R}^m$, a vector $\underline{c} \in \Lambda$ such that

$$\|\underline{x} - \underline{c}\| \leq \|\underline{x} - \underline{v}\| \qquad \text{for all } \underline{v} \in \Lambda$$

where $\|\cdot\|$ denotes the Euclidean norm.

The closest lattice point problem has applications in various fields, including number theory, cryptography, and communication theory [1]. Communication theory applications include quantization, multi-input multi-output (MIMO) decoding, and lattice coding for the Gaussian channel. Examples of such coding schemes are the recently introduced low-density lattice codes, which are based on lattices whose generator matrix has a sparse inverse [2], and signal codes, which are based on lattices with a Toeplitz generator matrix [3]. The computational complexity of the general closest point problem is known to be exponential with the lattice dimension n [4], [5].

Most of the existing methods that are used to solve the problem try to search the lattice points in an efficient manner. The commonly used sphere decoder, which is based on the Pohst enumeration [6], finds all the lattice points that lie within a

*Received by the editors November 30, 2006; accepted for publication (in revised form) December 7, 2008; published electronically April 9, 2009. A partial version of this paper appeared in *Proceedings of the* 2007 *IEEE International Symposium on Information Theory (ISIT)*, Nice, France, 2007.
http://www.siam.org/journals/sidma/23-2/67636.html

†Department of Electrical Engineering - Systems, Tel-Aviv University, Tel-Aviv, Israel (naftalis@eng.tau.ac.il, meir@eng.tau.ac.il, oshalvi@yahoo.com).

hypersphere of a given radius around the input point, where more efficient variants adapt the search radius [1] and optimize the search order [7]. A thorough explanation of the sphere decoder and its variants can be found in [1], and its outline is summarized in the appendix. The complexity of the sphere decoder can be further reduced by using algorithms that give approximate solutions [8], [9], [10]. In some applications, the complexity can be reduced by assuming specific lattice structures [2], [3].

In this paper we shall address only the problem of finding the *exact* solution for general lattices and present a new algorithm, called "the lattice slicer." This algorithm breaks the complex n-dimensional problem to a series of one-dimensional simple problems. In each such problem, the Euclidean space is divided into parallel layers, and the algorithm has to decide to which layer the input point belongs. This basic operation is referred to as "slicing." Using the results of the slicing steps, the algorithm can find the Voronoi cell of the lattice that contains the input point. In order to perform the slicing operations, a computationally intensive preprocessing stage is required. Therefore, the proposed algorithm is mainly suitable for cases where many closest lattice point problems have to be solved for the same lattice.

The computational complexity of the slicer algorithm is comparable to that of the sphere decoder, but it is significantly less sensitive to the eigenvalue spread of the lattice generator matrix $\boldsymbol{G}$. This is an important advantage in applications where the coefficients of $\boldsymbol{G}$ cannot be designed in advance or may even be random (e.g., MIMO decoding).

The outline of this work is as follows. First, section 2 introduces some basic definitions and properties of lattices. Section 3 presents the proposed lattice slicer algorithm, followed by convergence analysis in section 4 and discussion of the computational complexity in section 5. Section 6 presents a coordinate search interpretation of the lattice slicer, followed by a description of the preprocessing stage of the algorithm in section 7. Finally, simulation results are presented in section 8.

2. Basic definitions and notations. The *dot product* of two vectors $\underline{a}, \underline{b} \in \mathbb{R}^m$ is defined as $\underline{a} \cdot \underline{b} \triangleq \sum_{k=0}^{m-1} a_k b_k$. The Euclidean norm of $\underline{a} \in \mathbb{R}^m$ therefore equals $\|\underline{a}\| = \sqrt{\underline{a} \cdot \underline{a}}$.

The *Voronoi region* of a lattice point is the set of all vectors in $\mathbb{R}^m$ for which this point is the closest lattice point, namely,

$$\Omega(\Lambda, \underline{c}) \triangleq \{\underline{x} \in \mathbb{R}^m : \|\underline{x} - \underline{c}\| \leq \|\underline{x} - \underline{v}\| \qquad \forall \underline{v} \in \Lambda\},$$

where $\underline{c} \in \Lambda$. It is known [11] that the Voronoi regions $\Omega(\Lambda, \underline{c})$ are convex polytopes, that they are symmetrical with respect to reflection in $\underline{c}$, and that they are translations of $\Omega(\Lambda, \underline{0})$, where $\underline{0}$ is the origin of $\mathbb{R}^m$.

A *facet* is an $(m-1)$-dimensional face of an m-dimensional polytope.

The Voronoi region is uniquely determined by the *Voronoi relevant vectors*. These vectors lie opposite to the facets of the Voronoi region polytope and are the reflections of the origin in these facets. Therefore, each Voronoi relevant vector $\underline{v}$ defines a facet of $\Omega(\Lambda, \underline{0})$, where this facet is perpendicular to the Voronoi relevant vector $\underline{v}$ and intersects it in its midpoint $\frac{1}{2}\underline{v}$. A set of Voronoi relevant vectors is defined as a minimal set $N(\Lambda) \subseteq \Lambda$ for which

$$\Omega(\Lambda, \underline{0}) = \{\underline{x} \in \mathbb{R}^m : \|\underline{x}\| \leq \|\underline{x} - \underline{v}\| \qquad \forall \underline{v} \in N(\Lambda)\}.$$

A lattice of dimension n can have at most $2^{n+1} - 2$ Voronoi relevant vectors. This number is attained with probability 1 by a lattice whose basis is chosen at random from a continuous distribution (see [1] and references therein).

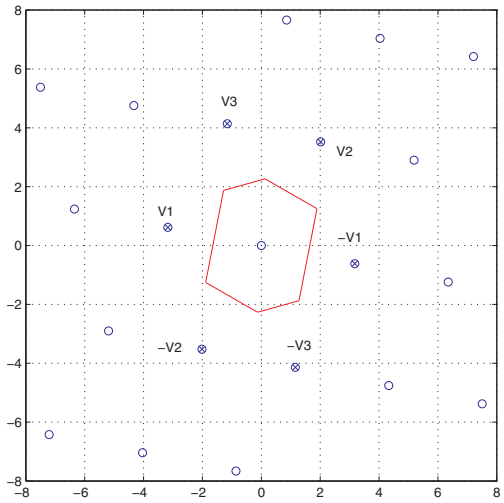

FIG. 1. *An example showing the Voronoi region of the origin and the Voronoi relevant vectors for a two-dimensional lattice. The lattice points are marked with a circle, and the Voronoi relevant vectors are marked with a "⊗." There are* 6 *Voronoi relevant vectors, denoted by* $\boldsymbol{v}_1, -\boldsymbol{v}_1, \boldsymbol{v}_2, -\boldsymbol{v}_2, \boldsymbol{v}_3$, *and* $-\boldsymbol{v}_3$.

Due to the symmetry of the Voronoi region, if $\underline{v}$ is a Voronoi relevant vector, then $-\underline{v}$ is also a Voronoi relevant vector. It is therefore convenient to define a "compressed" set that contains only a single representative from each pair $\{\underline{v}, -\underline{v}\}$. A *one-sided set of Voronoi relevant vectors* is thus defined as a minimal set $N'(\Lambda) \subseteq \Lambda$ for which

$$\Omega(\Lambda, \underline{0}) = \{\underline{x} \in \mathbb{R}^m : \|\underline{x}\| \leq \|\underline{x} - \underline{v}\|, \|\underline{x}\| \leq \|\underline{x} + \underline{v}\| \, \forall \underline{v} \in N'(\Lambda)\}.$$

Evidently, for a lattice of dimension n, the set of one-sided Voronoi relevant vectors can contain at most $2^n - 1$ vectors.

As an illustration, Figure 1 shows the Voronoi region of the origin and the Voronoi relevant vectors for a two-dimensional lattice.

3. The lattice slicer.

3.1. General description and some preliminaries. We shall now present the proposed algorithm—the lattice slicer. Finding the closest lattice point to $\underline{x} \in \mathbb{R}^m$ for a lattice Λ is equivalent to finding a lattice point $\underline{c} \in \Lambda$ such that the error $\underline{e} = \underline{c} - \underline{x}$ is inside the Voronoi region of the origin $\Omega(\Lambda, \underline{0})$. The lattice slicer algorithm starts with $\underline{c} = \underline{0}$ and iteratively updates $\underline{c}$ such that the resulting error vector $\underline{e}$ will finally reach $\Omega(\Lambda, \underline{0})$. In each step, $\underline{e}$ is checked against a pair of facets of $\Omega(\Lambda, \underline{0})$, defined by a single-sided Voronoi relevant vector, and $\underline{c}$ is updated such that the resulting $\underline{e}$ will be at the correct side of these facets. The algorithm terminates when $\underline{e}$ is at the correct side of all the facets of $\Omega(\Lambda, \underline{0})$.

Before proceeding to the detailed algorithm description, we need to prove some basic results. The following lemma gives an alternative definition to the one-sided set of Voronoi relevant vectors, where the conditions involve inner products instead of distances.

LEMMA 1. *Let* Λ *be a lattice in* $\mathbb{R}^m$. *Denote the Voronoi region of the origin by* $\Omega(\Lambda, \mathbf{0})$. *A one-sided set of Voronoi relevant vectors can be alternatively defined as a*

minimal set $N'(\Lambda) \subseteq \Lambda$ *for which*

$$\Omega(\Lambda, \underline{0}) = \left\{ \underline{x} \in \mathbb{R}^m : |\underline{x} \cdot \underline{v}| \leq \frac{1}{2} \|\underline{v}\|^2 \qquad \forall \underline{v} \in N'(\Lambda) \right\}.$$

Proof. According to the definition of the one-sided set of Voronoi relevant vectors, $\underline{x} \in \Omega(\Lambda, \underline{0})$ if and only if $\|\underline{x}\|^2 \leq \|\underline{x} - \underline{v}\|^2$ and $\|\underline{x}\|^2 \leq \|\underline{x} + \underline{v}\|^2$ for all $\underline{v} \in N'(\Lambda)$. Now, $\|\underline{x}\|^2 \leq \|\underline{x} - \underline{v}\|^2$ is equivalent to $\|\underline{x}\|^2 \leq \|\underline{x}\|^2 + \|\underline{v}\|^2 - 2\underline{x} \cdot \underline{v}$, which yields $\underline{x} \cdot \underline{v} \leq \frac{1}{2} \|\underline{v}\|^2$. In the same manner, $\|\underline{x}\|^2 \leq \|\underline{x} + \underline{v}\|^2$ is equivalent to $\underline{x} \cdot \underline{v} \geq -\frac{1}{2} \|\underline{v}\|^2$, so the pair of conditions $\|\underline{x}\|^2 \leq \|\underline{x} - \underline{v}\|^2$ and $\|\underline{x}\|^2 \leq \|\underline{x} + \underline{v}\|^2$ in the definition of the one-sided set of Voronoi relevant vectors can be replaced by the single condition $|\underline{x} \cdot \underline{v}| \leq \frac{1}{2} \|\underline{v}\|^2$, and the lemma is proved. ∎

Lemma 1 can be given a simple geometric interpretation. The condition $|\underline{x} \cdot \underline{v}| < \frac{1}{2} \|\underline{v}\|^2$ means that the length of the projection of $\underline{x}$ along the direction of $\underline{v}$ is less than $\frac{1}{2} \|\underline{v}\|$. This means that $\underline{x}$ is located between the two hyperplanes that are perpendicular to $\underline{v}$ and intersect it at $\frac{1}{2}\underline{v}$ and $-\frac{1}{2}\underline{v}$, respectively. Therefore, if this condition is satisfied for all the Voronoi relevant vectors, then $\underline{x}$ is at the "correct" side of all the facets that define the Voronoi region of the origin and is therefore inside this Voronoi region.

In what follows, we shall use rounding operations, where the basic rounding operation is defined as follows.

DEFINITION 1. *For every* $r \in \mathbb{R}$, *round*(r) *is defined as the closest integer to* r, *i.e., the integer* $k \in \mathbb{Z}$ *that minimizes* $|r - k|$. *If* r *is the midpoint between two consecutive integers, then the integer with smaller absolute value is chosen.*

Note that this definition is slightly different from the commonly used rounding operation, where a midpoint between consecutive integers is rounded to the integer with larger absolute value. The need for this modification will be explained later.

Lemma 1 imposed a set of conditions on a vector that lies inside the Voronoi region of the origin, where each condition involved one of the Voronoi relevant vectors. The next lemma suggests a way to modify a given vector if one of these conditions is not fulfilled.

LEMMA 2. *Let* $\underline{e}, \underline{v} \in \mathbb{R}^m$. *Define* $\tilde{\underline{e}} = \underline{e} - k\underline{v}$, *where*

$$k = round\left(\frac{\underline{e} \cdot \underline{v}}{\|\underline{v}\|^2} \right) \tag{1}$$

and where round$(\cdot)$ *is as in Definition* 1. *Then,* $|\tilde{\underline{e}} \cdot \underline{v}| \leq \frac{1}{2} \|\underline{v}\|^2$.

Proof. According to the definition of k, we can write $\underline{e} \cdot \underline{v} = \|\underline{v}\|^2 (k + \epsilon)$, where $|\epsilon| \leq \frac{1}{2}$ and $k \in \mathbb{Z}$. We then have

$$|\tilde{\underline{e}} \cdot \underline{v}| = |(\underline{e} - k\underline{v}) \cdot \underline{v}| = \left| \underline{e} \cdot \underline{v} - k \|\underline{v}\|^2 \right| = \left| \epsilon \|\underline{v}\|^2 \right| \leq \frac{1}{2} \|\underline{v}\|^2 \tag{2}$$

as desired. ∎

Lemma 2 has a geometric interpretation: If k was calculated without the rounding operation, $k\underline{v}$ would be the projection of $\underline{e}$ on $\underline{v}$. Subtracting this projection from $\underline{e}$ would result in a vector which is orthogonal to $\underline{v}$, thus minimizing $|\tilde{\underline{e}} \cdot \underline{v}|$. However, due to the rounding operation, $\tilde{\underline{e}}$ will be only "approximately orthogonal" to $\underline{v}$.

The geometric interpretation of Lemma 2 is illustrated in Figure 2, which shows the two-dimensional lattice of Figure 1. The figure shows the hyperplanes (for this

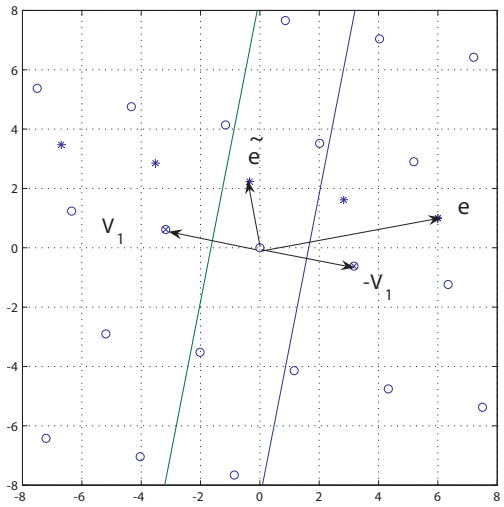

FIG. 2. *Example of the modification step of Lemma* 2.

case, straight lines) that intersect the Voronoi relevant vectors $\underline{v}_1, -\underline{v}_1$ at $\frac{1}{2}\underline{v}_1, -\frac{1}{2}\underline{v}_1$, respectively. The vector $\underline{e}$ lies outside the region that is bounded by the two straight lines, and we want to use Lemma 2 in order to insert it into this region. The stars mark possible locations for $\tilde{\underline{e}} = \underline{e} - k\underline{v}_1$ for various values of the integer k. Using (1), we get $k = -2$, and the resulting $\tilde{\underline{e}}$ is marked in the figure. It can be seen that this is the only value of k that puts $\tilde{\underline{e}}$ inside the desired region. Finding k is actually a slicing operation: The plane is divided into slices (or layers) whose boundaries are straight lines. These lines are orthogonal to the Voronoi relevant vector $\underline{v}_1$ and are spaced $\underline{v}_1$ apart from each other. The integer k actually denotes to which of these slices the vector $\underline{e}$ belongs.

Finally, the following lemma suggests a condition on the norms of two vectors that guarantees that their dot product satisfies the condition of Lemma 1.

LEMMA 3. *Let* $\underline{x}, \underline{v} \in \mathbb{R}^m$. *If* $\|\underline{x}\|^2 < \frac{1}{4}\|\underline{v}\|^2$, *then* $|\underline{x} \cdot \underline{v}| < \frac{1}{2}\|\underline{v}\|^2$.

Proof. $\|\underline{x}\|^2 < \frac{1}{4}\|\underline{v}\|^2$ implies that $\|\underline{x}\| < \frac{1}{2}\|\underline{v}\|$. Using the Cauchy–Schwarz inequality, we get

$$|\underline{x} \cdot \underline{v}| \leq \|\underline{x}\| \, \|\underline{v}\| < \frac{1}{2}\|\underline{v}\|^2$$

which completes the proof. ∎

We are now ready to formulate the lattice slicer algorithm.

3.2. Algorithm description. Given a vector $\underline{x} \in \mathbb{R}^m$ and a lattice Λ, we would like to find the closest lattice point $\underline{c}$ to $\underline{x}$ in an iterative manner. We initialize $\underline{c} = \underline{0}$. At each iteration, we check the error vector $\underline{e} = \underline{c} - \underline{x}$. If this error vector is inside the Voronoi region of the origin, then we know that $\underline{c}$ is the closest lattice point to $\underline{x}$. Now, from Lemma 1 we see that in order for $\underline{e}$ to be inside the Voronoi region of the origin, the absolute value of the dot product of this error vector with each one-sided Voronoi relevant vector should be smaller than half the norm of this Voronoi relevant vector. On the other hand, we see from Lemma 2 that if this condition is not fulfilled for a specific Voronoi relevant vector, then it can be fulfilled by modifying $\underline{e}$, where this modification is equivalent to adding a lattice vector to $\underline{c}$. Then, we can simply go

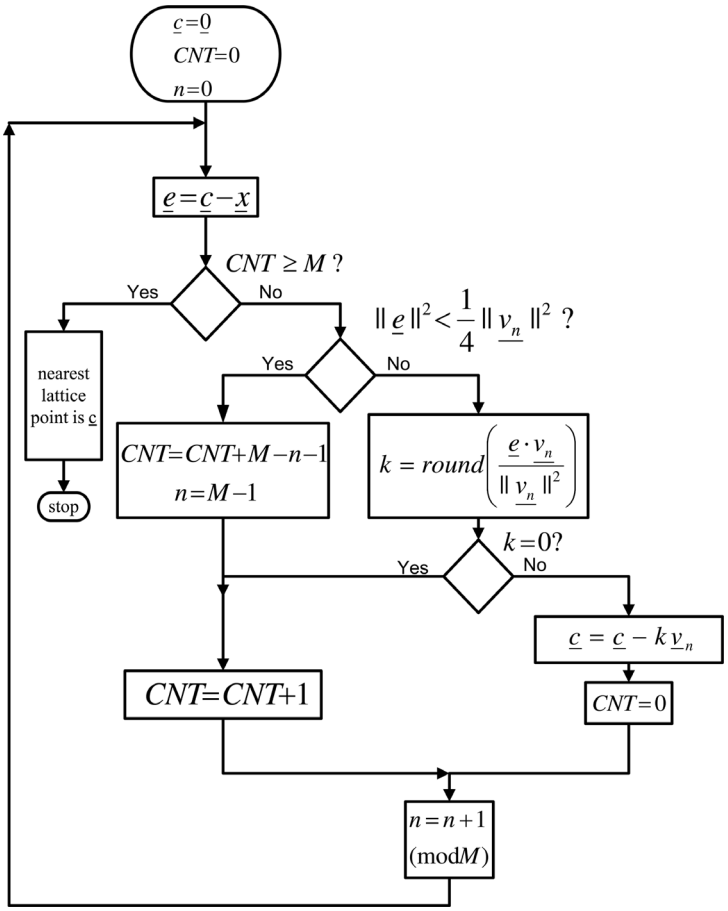

FIG. 3. *Flowchart of the lattice slicer algorithm.*

over the list of the one-sided Voronoi relevant vectors (assume that there are M such vectors). For each vector, if $\underline{e}$ does not satisfy the requirement of Lemma 1, then $\underline{c}$ is modified according to Lemma 2 such that the condition will be satisfied.

Using Lemma 3, there is no need to check the condition for Voronoi relevant vectors whose norm is more than twice the norm of the error. The list of Voronoi relevant vectors is assumed to be sorted by ascending norm, so the algorithm can skip to the end of the list if it reaches a vector that is longer than twice the current error vector.

After satisfying the condition for a specific Voronoi relevant vector, subsequent modifications due to other vectors may cause this condition to be unfulfilled again. Therefore, after finishing all the M Voronoi relevant vectors, the algorithm starts again from the first vector, and so on. Every check against a Voronoi relevant vector will be referred to as a "step," where every pass on the whole list of M vectors will be referred to as an iteration. If no change has been required for M consecutive steps, then the error finally reached the Voronoi region of the origin and the algorithm can be terminated.

The flowchart of the suggested algorithm is shown in Figure 3. The inputs to the algorithm are the vector $\underline{x} \in \mathbb{R}^m$ and the list of M one-sided Voronoi relevant vectors of the lattice Λ, denoted by $\underline{v}_0, \underline{v}_1, \ldots, \underline{v}_{M-1}$. It is assumed that this list was prepared

at the preprocessing stage of the algorithm (see section 7). It is also assumed that the list is sorted by ascending Euclidean norm, i.e., $\|\underline{v}_0\| \le \|\underline{v}_1\| \le \cdots \le \|\underline{v}_{M-1}\|$. The output of the algorithm is the vector $\underline{c}$ that holds the closest lattice point to $\underline{x}$. The algorithm uses the counter n, which points to the current Voronoi relevant vector, and the counter CNT which counts how many consecutive steps have passed without changing the vector $\underline{c}$. If the norm of the Voronoi relevant vector is larger than twice the norm of the current error vector, the algorithm jumps to the end of the list by updating n and CNT accordingly. If CNT reaches M, the error vector satisfies the conditions of Lemma 1 so the algorithm can be terminated.

Note that the calculation of k in Lemma 2 uses the $round(\cdot)$ operation of Definition 1, which takes special care of midpoints between integers. This is needed for singular situations where $\underline{e}$ lies exactly on a facet of the Voronoi region of the origin. In such a case, $\left|\frac{\underline{e}\cdot\underline{v}}{\|\underline{v}\|^2}\right| = \frac{1}{2}$ for a specific Voronoi relevant vector $\underline{v}$. Conventional rounding would round $\frac{1}{2}$ to 1 and $-\frac{1}{2}$ to -1, which may result in endless "ping-pong" between two facets. The modified rounding prevents this situation (see the convergence analysis in section 4).

Finally, it comes out that the computational complexity can be further reduced by initializing $\underline{c}$ to a better starting point, such as the rounded least squares solution $\underline{c} = round\left((\boldsymbol{G}'\boldsymbol{G})^{-1}\boldsymbol{G}'\underline{x}\right)$, where $\boldsymbol{G}$ is the generator matrix of the lattice and $\boldsymbol{G}'$ denotes the transpose of $\boldsymbol{G}$. Such initialization causes the error norm to start with a smaller value, so the algorithm can terminate earlier.

The operation of the algorithm is illustrated in Figure 4 for the two-dimensional lattice of Figure 1. The initial error is outside the Voronoi region of the origin. At the first step, the error is entered between the two facets that are perpendicular to $\underline{v}_1$. At the second step, the resulting error is entered between the two facets that are perpendicular to $\underline{v}_2$. Note that this error is no longer between the two facets that are perpendicular to $\underline{v}_1$. The process goes on with $\underline{v}_3$. Then, the algorithm starts again with $\underline{v}_1$. Continuing with $\underline{v}_2$, the error is finally brought inside the Voronoi region of the origin and the algorithm can terminate.

4. Convergence. In this section we shall show that the lattice slicer algorithm always converges to the closest lattice point within a finite number of steps. We shall first show that the norm of the error vector strictly decreases whenever $\underline{c}$ is changed.

LEMMA 4. *Let* $\underline{e}, \underline{v} \in \mathbb{R}^m$. *Define* $\tilde{\underline{e}} = \underline{e} - k\underline{v}$, *where* $k = round(\frac{\underline{e}\cdot\underline{v}}{\|\underline{v}\|^2})$ *and* $round(\cdot)$ *is as in Definition* 1. *Then,* $k \neq 0$ *implies* $\|\tilde{\underline{e}}\| < \|\underline{e}\|$.

Proof. Substituting for $\tilde{\underline{e}}$, we have

$$\|\tilde{\underline{e}}\|^2 - \|\underline{e}\|^2 = (\underline{e} - k\underline{v}) \cdot (\underline{e} - k\underline{v}) - \|\underline{e}\|^2 = -2k\underline{e} \cdot \underline{v} + k^2 \|\underline{v}\|^2 . \tag{3}$$

According to the definition of k, we can write $\underline{e} \cdot \underline{v} = \|\underline{v}\|^2 (k + \epsilon)$, where $|\epsilon| \le \frac{1}{2}$ and $k \in \mathbb{Z}$. Substituting in (3), we get

$$\|\tilde{\underline{e}}\|^2 - \|\underline{e}\|^2 = -2k(k+\epsilon) \|\underline{v}\|^2 + k^2 \|\underline{v}\|^2 = - \|\underline{v}\|^2 k(k+2\epsilon). \tag{4}$$

Since $|\epsilon| \le \frac{1}{2}$ and $k \in \mathbb{Z}$, the expression $k(k+2\epsilon)$ is strictly positive for $|k| \ge 2$, resulting in $\|\tilde{\underline{e}}\|^2 < \|\underline{e}\|^2$, as desired. We still have to show that $k(k+2\epsilon)$ is strictly positive for $k = \pm 1$. Assuming that $k = 1$, the only way for $k(k+2\epsilon)$ to be nonpositive is if $\epsilon = -\frac{1}{2}$. However, Definition 1 does not allow us to have $\epsilon = -\frac{1}{2}$ for $k = 1$, since the value $\frac{1}{2}$ is rounded to 0 and not to 1. In the same manner, we cannot have $k = -1$ with $\epsilon = \frac{1}{2}$. Therefore, $k(k+2\epsilon)$ is strictly positive for all $k \neq 0$, resulting in $\|\tilde{\underline{e}}\|^2 < \|\underline{e}\|^2$ for $k \neq 0$, as desired. ∎

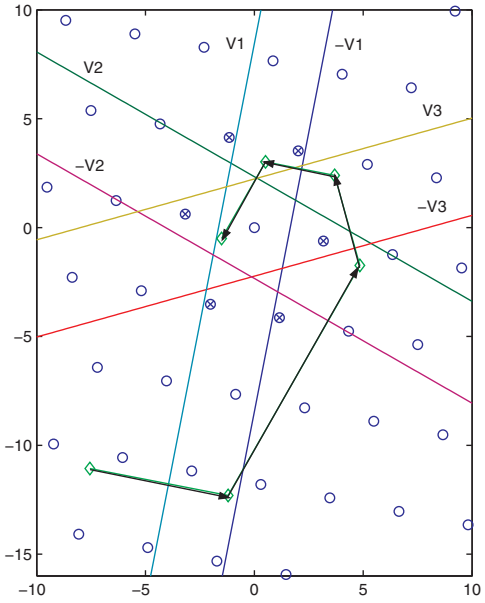

FIG. 4. *Convergence of the slicer algorithm. Every facet of the Voronoi region of the origin is marked with the name of the corresponding Voronoi relevant vector.*

Using Lemma 4, we can now prove the main theorem of this section.

THEOREM 1. *Given a lattice* Λ *and* $\underline{x} \in \mathbb{R}^m$*, the lattice slicer algorithm of Figure 3 always converges to the closest lattice point of* $\underline{x}$ *in a finite number of steps.*

Proof. The algorithm starts with $\underline{c} = \underline{0}$ and updates $\underline{c}$ along the iterations. Lemma 4 shows that whenever $\underline{c}$ is updated with $k \neq 0$, the norm of the error vector $\underline{e} = \underline{c} - \underline{x}$ strictly decreases. As a result, whenever $\underline{c}$ is changed, it cannot assume the same lattice point again. Now, the initial error norm is $\|\underline{e}\| = \|\underline{x}\|$, so the error norm is bounded above by $\|\underline{x}\|$ for all the iterations of the algorithm. This means that $\underline{c}$ always lies within a hypersphere of radius $\|\underline{x}\|$ around $\underline{x}$. Having a finite radius, this hypersphere contains a finite number of lattice points. As long as the algorithm does not terminate, it must update $\underline{c}$ with $k \neq 0$ at least once every M steps. Such an update changes $\underline{c}$ from one lattice point to another. Since the algorithm cannot return to the same lattice point twice and it can visit only a finite number of points, it must terminate. According to Lemma 1, when the algorithm terminates, the error vector $\underline{e}$ is inside the Voronoi region of the origin, so $\underline{c}$ must be the closest lattice point. That completes the proof of the theorem. ∎

Finally, the following lemma shows that the set of Voronoi relevant vectors is the minimal set that guarantees convergence to the closest lattice point.

LEMMA 5. *Consider the lattice slicer algorithm, and assume that the list of* M *Voronoi relevant vectors is replaced with a different list of* N *arbitrary lattice vectors. Then, the algorithm will converge to the closest lattice point for every* $\underline{x} \in \mathbb{R}^m$ *if and only if this list of* N *lattice vectors contains all the* M *Voronoi relevant vectors of the lattice.*

Proof. First, it can be seen from the proof of Theorem 1 that the algorithm terminates for any list of lattice points, since Lemma 4 holds for any $\underline{v}$. Now, assume that the list contains all the Voronoi relevant vectors. Then, according to Lemma 1, the resulting lattice point must be the closest lattice point. Assume now that a single

Voronoi relevant vector is missing from the list. According to the definition of the Voronoi relevant vectors, there exists a point $\underline{x}_0 \in \mathbb{R}^m$ which is outside the Voronoi region of the origin, but it satisfies the condition of Lemma 1 for all the vectors in the list. Therefore, if the algorithm runs with $\underline{x} = \underline{x}_0$, it will terminate after a single iteration and return the origin as the closest lattice point. However, $\underline{x}_0$ is outside the Voronoi region of the origin. Therefore, all the Voronoi relevant vectors must be included in the list to ensure convergence to the closest lattice point. □

5. Computational complexity. The most computationally intensive part of the algorithm is the calculation of k. This is the only computation that is performed every step and involves vector operations, since the other computations that involve vector operations can be performed only when $\underline{c}$ is changed. Also, the squared norms of the Voronoi relevant vectors, as well as their reciprocals, can be calculated in advance. The calculation of k involves a single dot product. Assuming an n-dimensional lattice in $\mathbb{R}^m$, this is an $O(m)$ operation. Since there are $O(2^n)$ Voronoi relevant vectors (see section 2), the complexity of a full iteration is $O(m2^n)$.

We would now like to bound the overall complexity of the algorithm. As shown in section 4, for an input point $\underline{x}$, the number of iterations is bounded above by the number of lattice points in a hypersphere of radius $\|\underline{x}\|$. The volume of the Voronoi region of a lattice with generator matrix $\boldsymbol{G}$ is $\sqrt{\det(\boldsymbol{G}'\boldsymbol{G})}$, where the volume of a hypersphere with radius $\|\underline{x}\|$ and dimension n is $\frac{\pi^{n/2}}{\Gamma(n/2+1)}\|\underline{x}\|^n$ [11]. However, we cannot simply divide the sphere volume by the Voronoi region volume, since a lattice point may be contained in the sphere where only part of its Voronoi region is contained in it. We can bound the number of lattice points from above by using a sphere with increased radius $\|\underline{x}\| + \Delta$ such that, for every lattice point which is contained in a sphere with radius $\|\underline{x}\|$, its whole Voronoi region will be contained in a sphere with radius $\|\underline{x}\| + \Delta$. It can be easily seen that a proper choice for Δ is the norm of the longest vector which is contained in the Voronoi region of the origin. Note that for this choice, Δ may depend, in general, on the lattice dimension. For example, for a rectangular lattice, whose generator matrix is the identity matrix of dimension n, the Voronoi region is a hypercube, whose longest vector has a norm proportional to $\sqrt{n}$. If the coordinates of the input point $\underline{x}$ are chosen randomly and independently, the average norm of $\underline{x}$ will also be proportional to $\sqrt{n}$. Therefore, the effect of Δ on the bound will become negligible only for input points which are far from the origin, i.e., $\|\underline{x}\| >> \Delta$.

An upper bound for the overall complexity of the algorithm can now be obtained by multiplying the bound on the number of lattice points in a sphere of radius $\|x\|$ by the complexity of a full iteration, yielding an upper bound on the complexity whose behavior is

$$O\left(\frac{m2^n\pi^{n/2}}{\Gamma(n/2+1)}\frac{(\|\underline{x}\|+\Delta)^n}{\sqrt{\det(\boldsymbol{G}'\boldsymbol{G})}}\right) \tag{5}$$

operations.

We would now like to develop a similar bound for the sphere decoder (see section 1 and the appendix). The sphere decoder uses an upper triangular representation of the lattice generator matrix in order to search sequentially for all the lattice points inside a sphere of radius d and dimension n. At the worst case, in order to reach all these points, the algorithm has to search all the points inside spheres with radius d and dimensions $1, 2, \ldots, n$ [12]. The search radius d must be larger or equal to the

norm of the closest lattice point to $\underline{x}$. Using the above arguments, we can derive an upper bound on the complexity of the sphere decoder, whose behavior is

$$O\left(\frac{1}{\sqrt{\det(\boldsymbol{G}'\boldsymbol{G})}}\sum_{k=1}^{n}\frac{\pi^{k/2}}{\Gamma(k/2+1)}(d+\Delta)^k\right) \tag{6}$$

operations.

Comparing (5) and (6), we can see that both expressions depend exponentially on the lattice dimension n, as expected from solutions to the closest lattice point problem. It can be easily seen that the summation of exponential terms in (6) will result in a different constant outside the exponential term but will not change the base of the exponential term itself. However, the base of the exponential terms is still different. In (5), we have an additional factor of 2^n, which potentially increases the exponential base by a factor of 2. On the other hand, in (5), the exponential base depends on $\|\underline{x}\|$ where in (6) it depends on the search radius d. In general, d has to be larger than $\|\underline{x}\|$, in order for the search sphere to contain the closest lattice point to $\underline{x}$. As $\|\underline{x}\|$ is not known in advance, even a larger search radius must be used in practice, resulting in an increase in the exponential base of (6). As described in the appendix, an alternative approach for the sphere decoder is to set a large initial search radius and then decrease it whenever a closer lattice point is found. For this case, the search radius changes with time, but the "effective" radius is still larger than $\|\underline{x}\|$, since most of the time the algorithm has to use a larger radius. Whether the search radius is fixed or adaptive, the effective radius will probably be less than $2\|\underline{x}\|$, so for large dimensions we should expect the asymptotic complexity of the sphere decoder to have a better exponential base than that of the lattice slicer, due to the additional $O(2^n)$ complexity factor of each lattice slicer iteration.

Note that we expect the average complexity of the lattice slicer to be much smaller than the above bound. The number of iterations will be much smaller than the number of lattice points in a sphere with radius $\|\underline{x}\|$, as the correction steps will move the error vector quickly close to the origin, and many times a single iteration will suffice. Also, the complexity per iteration will be much smaller than $O(m2^n)$, mainly due to the test of Lemma 3. Unless this test was used, a full iteration of $O(m2^n)$ operations would be needed for *every* input vector $\underline{x} \in \mathbb{R}^m$, since every Voronoi relevant vector has to be checked at least once. However, if $\underline{x}$ is relatively close to the closest lattice point, the required complexity is much less than a full iteration, as the test of Lemma 3 inhibits many checks. Although the worst-case complexity may still be $O(m2^n)$, using this test significantly improves the *average* complexity when the input vector is uniformly distributed in the Voronoi cell of the lattice (such as for quantization applications). Even larger improvement is expected when the input vector is a lattice point with additive Gaussian noise (such as for MIMO decoding or lattice coding for the Gaussian channel), since the initial error vector will be short with higher probability than for an input point which is uniformly distributed over the Voronoi cell.

The required storage of the algorithm is $O(m2^n)$, since all the Voronoi relevant vectors must be prepared in advance and stored during operation.

6. Coordinate search interpretation. We shall now show that the lattice slicer algorithm can be also interpreted as a coordinate search solution for an optimization problem. We shall first prove the following lemma, which is an extension of Lemma 4.

LEMMA 6. *Let* $\underline{e}, \underline{v} \in \mathbb{R}^m$. *Define* $\underline{\tilde{e}} = \underline{e} - k\underline{v}$, *where* $k \in \mathbb{Z}$. *Then, choosing* $k = k_0 \stackrel{\Delta}{=} round(\frac{\underline{e}\cdot\underline{v}}{\|\underline{v}\|^2})$, *where* $round(\cdot)$ *is defined in Definition* 1, *minimizes the Euclidean norm of* $\underline{\tilde{e}}$ *over all the possible choices of* k:

$$k_0 = \arg\min_{k\in\mathbb{Z}} \|\underline{e} - k\underline{v}\| . \tag{7}$$

Proof. We shall first show that an expression of the form $ak^2 + bk + c$, where $k \in \mathbb{Z}$ and $a > 0$, is minimized for $k = round(\frac{-b}{2a})$. For $x \in \mathbb{R}$ and $a > 0$, the function $f(x) = ax^2 + bx + c$ has a single global minimum at $x_0 = \frac{-b}{2a}$, where it is monotonically increasing for $x > x_0$ and monotonically decreasing for $x < x_0$. Therefore, an expression of the form ak^2+bk+c, where $k \in \mathbb{Z}$, must have its minimum either at $\lfloor x_0 \rfloor$ (the largest integer smaller than or equal to x_0) or at $\lceil x_0 \rceil$ (the smallest integer larger than or equal to x_0). Now, the function $f(x) = ax^2+bx+c$ is symmetric about x_0 (i.e., $f(x_0 + x) = f(x_0 - x)$), so the minimum will be achieved at the closer point to x_0, i.e., at $round\left(\frac{-b}{2a}\right)$.

If we now substitute for $\tilde{e}$, we have

$$\|\underline{\tilde{e}}\|^2 = (\underline{e} - k\underline{v}) \cdot (\underline{e} - k\underline{v}) = \|\underline{e}\|^2 - 2k\underline{e} \cdot \underline{v} + k^2 \|\underline{v}\|^2 . \tag{8}$$

This is an expression of the form $ak^2 + bk + c$, so the minimum is achieved for $k = round(\frac{-b}{2a}) = round(\frac{\underline{e}\cdot\underline{v}}{\|\underline{v}\|^2})$. Note that using the $round(\cdot)$ function of Definition 1 instead of conventional rounding will affect only which point to choose in case of two minimum points with equal value. ∎

Lemma 6 shows that each step of the lattice slicer algorithm can be regarded as minimization of the error norm, where the minimization is done along the direction of a Voronoi relevant vector (with an integer coefficient). The Voronoi relevant vectors define a "bank of directions," and the algorithm tries to minimize the error norm along each direction. It terminates when it can no longer minimize along any of the directions. Since all the directions are lattice points and only integer coefficients are used, the resulting point must be a lattice point. Theorem 1 shows that choosing the bank of directions as the set of all the Voronoi relevant vectors ensures that this coordinate search converges to the global minimum, which is the closest lattice point. Furthermore, Lemma 5 shows that the set of all the Voronoi relevant vectors is the minimal set of directions that will assure convergence.

7. Preprocessing: Finding the Voronoi relevant vectors. The lattice slicer algorithm uses a list of all the Voronoi relevant vectors of the lattice, which should be computed at the preprocessing stage of the algorithm. An efficient algorithm for the computation of the Voronoi relevant vectors of a lattice Λ with generator matrix $\boldsymbol{G}$ is the *RelevantVectors* algorithm, proposed in [1]. First, the set of midpoints $\mathcal{M}(\boldsymbol{G})$ is defined as

$$\mathcal{M}(\boldsymbol{G}) \stackrel{\Delta}{=} \{\underline{s} = \boldsymbol{G}\underline{z} : \underline{z} \in \{0, 1/2\}^n - \{\underline{0}\}\} . \tag{9}$$

For each $\underline{s} \in \mathcal{M}(\boldsymbol{G})$, the algorithm finds all the closest lattice points to $\underline{s}$, using the *AllClosestPoints* algorithm, which is an enhanced sphere decoder (see detailed description in the appendix). Then, if there are exactly two closest lattice points $\underline{x}_1, \underline{x}_2$ (i.e., two points at the same distance from $\underline{s}$ such that this distance is strictly smaller than the distance of any other lattice point from $\underline{s}$), then $2(\underline{x}_1 - \underline{s})$ and $2(\underline{x}_2 - \underline{s})$ are Voronoi relevant vectors of Λ. If the one-sided set of Voronoi relevant vectors is

required, it suffices to take only one of these two lattice points. See [1] for a proof that this algorithm finds all the Voronoi relevant vectors.

Note that the computational complexity of this preprocessing stage, which requires $2^n - 1$ runs of the enhanced sphere decoder, is significantly higher than that of the lattice slicer itself. However, in many applications there is a need for solving several closest lattice points problems for the same lattice. In such cases, the preprocessing stage runs only once per many runs of the lattice slicer itself, so its complexity becomes negligible.

8. Simulation results. The lattice slicer algorithm was simulated with the following model. The elements of the $n \times n$ lattice generator matrix $\boldsymbol{G}$ were generated as independent Gaussian random variables with zero mean and unit variance, and then $\boldsymbol{G}$ was normalized to have a determinant of 1. The elements of the n-dimensional input vector $\underline{x}$ were generated as independent Gaussian random variables with zero mean and variance σ^2. For small σ, $\underline{x}$ belongs to the Voronoi region of the origin. If σ is significantly larger than the dimensions of the Voronoi region, $\underline{x}$ can be regarded as uniformly distributed in the Voronoi cell of the closest lattice point (which is not necessarily the origin).

Computational complexity is measured by the average number of floating point operations (flops). Multiplication, addition, subtraction, and comparison are regarded as a single flop. The division operation of (1) is assumed to be implemented with multiplication by the reciprocal of the denominator, which can be calculated in advance.

The lattice slicer is compared with the *AllClosestPoints* algorithm of [1], which is an enhanced sphere decoder (see detailed description in the appendix). No assumption is made on the range of the integer coordinates of the lattice point. The preprocessing stage of the sphere decoder includes only QR decomposition of $\boldsymbol{G}$ (no lattice reduction).

In all simulations, the lattice slicer is initialized with the rounded least squares solution $\underline{c} = round\left((\boldsymbol{G}'\boldsymbol{G})^{-1}\boldsymbol{G}'\underline{x}\right)$, where $\boldsymbol{G}$ is the generator matrix of the lattice, as described at the end of section 3.2.

Figure 5 shows the average computational complexity of the lattice slicer and the enhanced sphere decoder as a function of the input point standard deviation for a lattice dimension of $n = 8$. Every point in the graph was generated by averaging over 1000 randomly generated $\boldsymbol{G}$ matrices with 50 randomly generated instances of the input point $\underline{x}$ per each $\boldsymbol{G}$. It can be seen that the average complexity of the sphere

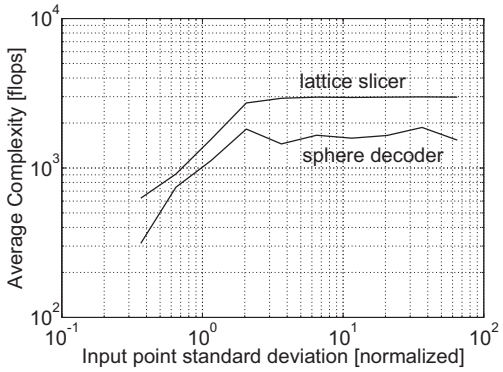

FIG. 5. *Average complexity vs. input point standard deviation, lattice dimension* = 8.

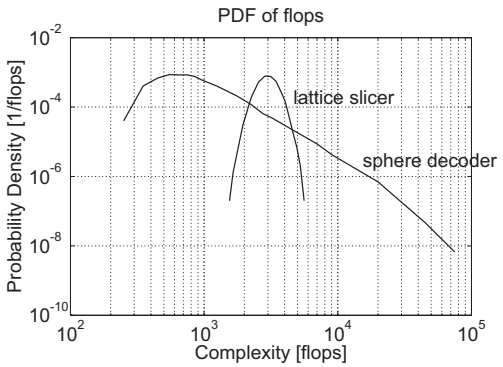

FIG. 6. *Probability density function (PDF) of the number of flops, lattice dimension* $= 8$*, input point standard deviation* $= 500$.

decoder is better by a factor of 2. However, the flops count is only a rough complexity measure, and it may significantly change for specific implementations. Therefore, the factor of 2 is small enough such that the algorithms can be regarded as having comparable complexity.

Note that for small input point standard deviations, the complexity increases with the standard deviation for both algorithms. The reason is that the input point is still in the Voronoi region of the origin, relatively close to the origin itself. When the standard deviation gets smaller, the lattice slicer can skip more Voronoi relevant vectors using the test of Lemma 3, and the sphere decoder reaches the correct point earlier with higher probability. When the input point standard deviation increases beyond some level, the curves flatten and complexity becomes constant, since the input point is now approximately uniformly distributed along the Voronoi region of the closest lattice point (which is not necessarily the origin).

Figure 6 shows the probability density function (PDF) of the computational complexity, as estimated from the histogram of the computational complexity of the lattice slicer and the enhanced sphere decoder. The lattice dimension was 8, and the input point standard deviation was 500, which is large enough to assume uniform distribution of the input point along the Voronoi cell of the closest lattice point. Each trial in the histogram corresponds to a different $\boldsymbol{G}$. Five-thousand $\boldsymbol{G}$ matrices were generated, and the complexity was averaged for each $\boldsymbol{G}$ over 50 instances of the input point. It can be seen that the two PDFs have different behavior: The PDF of the lattice slicer is concentrated around its peak and decays sharply, where the PDF of the sphere decoder has a heavy tail. This heavy tail causes the sphere decoder to have larger computational variability than the lattice slicer. A possible measure for the computational variability is the peak-to-average ratio of the computational complexity for a finite set of $\boldsymbol{G}$ matrices, defined as the ratio between the average complexity for the worst-case $\boldsymbol{G}$ in the set and the average complexity for the whole set. The sphere decoder's peak-to-average ratio for a set of 5,000 $\boldsymbol{G}$ matrices was measured to be larger by a factor of 20 than that of the lattice slicer.

The larger variability of the sphere decoder can be explained as follows. The complexity of the sphere decoder is larger for $\boldsymbol{G}$ matrices that are close to being singular, since such matrices tend to have elements with small magnitude on the diagonal of the corresponding upper triangular matrix. Such elements cause the sphere decoder to search a large list of values for the corresponding integers, resulting in large computational complexity peaks.

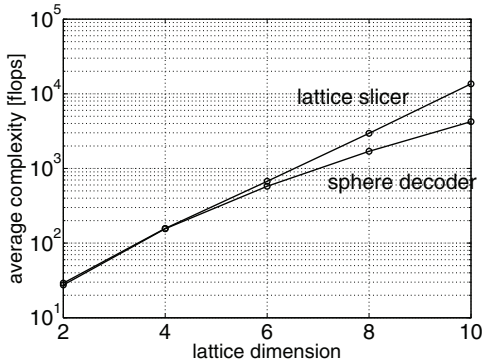

FIG. 7. *Average complexity vs. lattice dimension, input point standard deviation* = 500*, randomly generated lattices.*

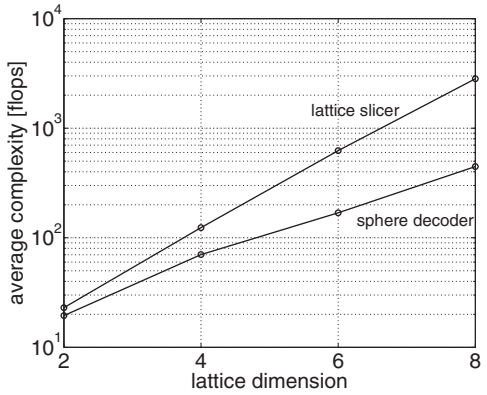

FIG. 8. *Average complexity vs. lattice dimension, input point standard deviation* = 500*, structured lattices.*

Figure 7 shows the average computational complexity of both algorithms as a function of the lattice dimension for randomly generated lattice generator matrices and input points (with the simulation conditions as in Figure 5). The input point standard deviation is 500, which is large enough such that the input point is uniformly distributed along the Voronoi cell of the closest lattice point. It can be seen that both algorithms have the same complexity for small dimensions, where the sphere decoder has an advantage for large dimensions (where this advantage increases as the dimension increases). This behavior is consistent with the upper bounds of section 5, which predict that for large dimensions the complexity of both algorithms will be exponential with the lattice dimension, but the exponential base for the lattice slicer will be larger.

So far we have examined the computational complexity for random lattice generator matrices. It is also interesting to compare the complexity of the algorithms for structured lattices. Figure 8 shows the computational complexity as a function of lattice dimension for the densest possible lattice packings in dimensions 2, 4, 6, and 8, which are the A_2, D_4, E_6, and E_8 lattices, respectively [11]. The generator matrix of each lattice was normalized to have a determinant of 1. For each lattice, 50,000 input points were generated and the complexity was averaged. It can be seen that the computational complexity of the lattice slicer in Figure 8 is approximately the same as in Figure 7. However, the complexity of the sphere decoder is much better. For

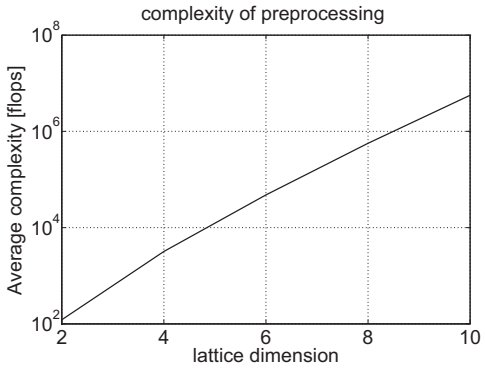

FIG. 9. *Complexity of the preprocessing stage vs. lattice dimension.*

example, for a lattice dimension of 8, the average complexity of the sphere decoder reduces from 2,000 to 450 flops. These results illustrate that the lattice slicer is almost insensitive to the lattice generator matrix, where the complexity of the sphere decoder is much worse for matrices which are close to being ill-conditioned. This dependency degrades the average complexity of the sphere decoder for randomly generated lattices and is also the cause for its large computational variability, as shown in Figure 6. Therefore, the lattice slicer can have advantages over the sphere decoder when the lattice is generated randomly (e.g., in MIMO communications), but the sphere decoder is superior for fixed, structured and well-conditioned lattices.

All the above simulations measure only the complexity of the lattice slicer itself, ignoring the preprocessing, since we assume that many closest lattice point problems are solved for the same lattice. However, it is of interest to evaluate the complexity of the preprocessing stage as well. The preprocessing stage finds the Voronoi relevant vectors using the *RelevantVectors* algorithm of [1], described in section 7. This algorithm calls the *AllClosestPoints* enhanced sphere decoder subroutine, described in the appendix. In each call, it is sufficient to use a search radius of half the norm of the longest Voronoi relevant vector (see section 7). However, the norms of the Voronoi relevant vectors are not known in advance. It came out by trial and error that, for this simulation setting, efficient implementation was achieved by using a search radius of 1.6 in each call to the enhanced sphere decoder. If no point was found, the search radius was multiplied by 1.25, and so on, until at least one point was found.

Figure 9 shows the average complexity of the preprocessing stage as a function of lattice dimension. One-thousand random $\boldsymbol{G}$ matrices were generated for each dimension, with 50 randomly generated input points for each $\boldsymbol{G}$, and the complexity was averaged over all of them. Comparing Figures 9 and 7, it can be seen that the complexity of the preprocessing stage is approximately 3, 10, and 100 times the average complexity of the lattice slicer itself for lattice dimensions of 2, 4, and 8, respectively.

9. Conclusion. A novel algorithm was proposed for the closest lattice point problem. The algorithm uses the list of Voronoi relevant vectors of the lattice in order to break the complex n-dimensional problem into a series of one-dimensional simple slicing problems. It was shown that the algorithm is guaranteed to converge in a finite number of steps and that it can be interpreted as a coordinate search solution for an optimization problem. Simulation results were presented, showing that the lattice slicer algorithm has average complexity which is comparable to that

of the enhanced sphere decoder, but its computational variability is smaller. The lattice slicer algorithm is an additional tool in the "toolkit" of closest lattice point algorithms, giving additional insight to the closest lattice point problem.

Topics for further research include enhancing the speed of convergence by optimizing the order of the list of Voronoi relevant vectors. For example, the list can be divided into groups, where the vectors in each group are "nearly parallel" to each other such that starting with a single representative from each group will decrease the error norm in a fast rate, and then most of the other group elements could be skipped. Another possible direction is finding an approximate solution by using only part of the Voronoi relevant vectors, which may be an attractive option for lattices with large dimension. Also, convergence speed may be improved by iterating the short vectors several times before using (if needed) the longer vectors.

Appendix A. The sphere decoder and its variants. This appendix describes the outline of the sphere decoder and its variants. See [1] for a thorough explanation.

Given a lattice Λ with an $m \times n$ generator matrix $\boldsymbol{G}$, we want to find the closest lattice point to a given vector $\underline{r} \in \mathbb{R}^m$. We can then write $\underline{r} = \boldsymbol{G}\underline{u} + \underline{w}$, where $\underline{u} \in \mathbb{Z}^n$ and $\underline{w} \in \mathbb{R}^m$ is the noise vector whose norm should be minimized. The popular sphere decoder, which searches all the lattice points within a sphere of a given radius ρ around $\underline{r}$, is based on the Pohst enumeration [6]. Using QR factorization of the matrix $\boldsymbol{G}$, the problem is transformed to an equivalent problem but with an upper triangular matrix: $\tilde{\underline{r}} = \boldsymbol{R}\underline{u} + \tilde{\underline{w}}$, where $\boldsymbol{R}$ is an $n \times n$ upper triangular matrix with positive diagonal elements (assuming $n \leq m$). The sphere decoder then starts from the bottom nth row and climbs up through a search tree in a depth first search manner. At row $n-k$, given a sequence of candidates for $u_n, u_{n-1}, \ldots, u_{n-k+1}$, a candidate list is prepared for u_{n-k} including all values such that $\sum_{i=0}^{k} \left|\tilde{r}_{n-i} - \sum_{m=n-i}^{n} R_{n-i,m} u_m\right|^2 < \rho^2$, i.e., the contribution of the bottom $k+1$ coordinates to the squared distance of the tested lattice point from $\underline{r}$ is still less than ρ^2. Each member of this candidate list corresponds to a tree node at layer $n-k$. Whenever the algorithm reaches the first row (layer 1), the distance of the resulting lattice point from the received vector $\underline{r}$ is recorded and the closest point is updated. If no point is found, the search radius is increased and the algorithm starts again.

The basic Pohst enumeration can be improved by dynamically updating the search radius to the distance of $\underline{r}$ from the closest point found so far. A further improvement is the Schnorr–Euchner strategy [7], which is equivalent to sorting the candidate list for u_{n-k} by increasing value of the score increment $\Delta score_{n-k} = \left|\tilde{r}_{n-k} - \sum_{m=n-k}^{n} R_{n-k,m} u_m\right|^2$. The *AllClosestPoints* algorithm of [1], which is used in this paper for the preprocessing stage of the lattice slicer, as well as for complexity comparison of the sphere decoder with the lattice slicer, is based on the basic sphere decoder with these two improvements.

Acknowledgments. Support and interesting discussions with Ehud Weinstein are gratefully acknowledged. The authors would also like to thank the anonymous reviewers for their thorough review and valuable comments.

REFERENCES

[1] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger, *Closest point search in lattices*, IEEE Trans. Inform. Theory, 48 (2002), pp. 2201–2214.

[2] N. Sommer, M. Feder, and O. Shalvi, *Low density lattice codes*, IEEE Trans. Inform. Theory, 54 (2008), pp. 1561–1585.

[3] O. Shalvi, N. Sommer, and M. Feder, *Signal Codes*, in Proceedings of the IEEE Information Theory Workshop, Paris, 2003, pp. 332–336.
[4] P. van Emde Boas, *Another NP-complete Partition Problem and the Complexity of Computing Short Vectors in a Lattice*, Report 81-04, Mathematisch Instituut, Amsterdam, The Netherlands, 1981.
[5] D. Micciancio, *The hardness of the closest vector problem with preprocessing*, IEEE Trans. Inform. Theory, 47 (2001), pp. 1212–1215.
[6] U. Fincke and M. Pohst, *Improved methods for calculating vectors of short length in a lattice, including a complexity analysis*, Math. Comp., 44 (1985), pp. 463–471.
[7] C. P. Schnorr and M. Euchner, *Lattice basis reduction: Improved practical algorithms and solving subset sum problems*, Math. Program., 66 (1994), pp. 181–191.
[8] N. Sommer, M. Feder, and O. Shalvi, *Closest point search in lattices using sequential decoding*, in Proceedings of the IEEE International Symposium on Information Theory (ISIT), Adelaide, Australia, 2005, pp. 1053–1057.
[9] R. Gowaikar and B. Hassibi, *Efficient maximum-likelihood decoding via statistical pruning*, IEEE Trans. Inform. Theory, submitted.
[10] M. O. Damen, H. El Gamal, and G. Caire, *On maximum-likelihood decoding and the search of the closest lattice point*, IEEE Trans. Inform. Theory, 49 (2003), pp. 2389–2402.
[11] J. H. Conway and N. J. A. Sloane, *Sphere Packings, Lattices and Groups*, 3rd ed., Springer-Verlag, New York, 1999.
[12] B. Hassibi and H. Vikalo, *On the sphere-decoding algorithm* I. *Expected complexity*, IEEE Trans. Signal Process., 53 (2005), pp. 2806–2818.