CHALMERS LINDHOLMEN

Closest Point Search in Lattices

ERIK AGRELL
THOMAS ERIKSSON
ALEXANDER VARDY
KENNETH ZEGER

CHALMERS LINDHOLMEN UNIVERSITY COLLEGE
Göteborg, Sweden 2000

Report no. 1, Feb. 2000

REPORT No. 1

Closest Point Search in Lattices

ERIK AGRELL
Department of Electrical and Computer Engineering
Chalmers Lindholmen University College
Göteborg, Sweden

THOMAS ERIKSSON
Department of Signals and Systems
Chalmers University of Technology
Göteborg, Sweden

ALEXANDER VARDY and KENNETH ZEGER
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, CA, USA

CHALMERS LINDHOLMEN UNIVERSITY COLLEGE
Göteborg, Sweden, Feb. 2000

Closest Point Search in Lattices

ERIK AGRELL
THOMAS ERIKSSON
ALEXANDER VARDY
KENNETH ZEGER

Report – Chalmers Lindholmen University College
Report no. 1
ISSN 1404-5001

Chalmers Lindholmen University College
P.O. Box 8873
SE-402 72 Göteborg
Sweden
Telephone + 46 31 772 1000

Chalmers Lindholmen University College
Göteborg, Sweden, Feb. 2000

Closest Point Search in Lattices

E. AGRELL, T. ERIKSSON, A. VARDY, and K. ZEGER

SUMMARY

In this semi-tutorial report, algorithms for finding the closest point in a lattice are studied. The complexity of different strategies is compared, both theoretically and by experiments. A complete implementation of an efficient closest-point algorithm is given, together with straightforward modifications of the algorithm to solve a number of related search problems for lattices, such as finding a shortest vector, determining the kissing number, computing the Voronoi-relevant vectors, or finding a Korkine-Zolotareff reduced basis.

Keywords: Closest-point search, kissing number, Korkine-Zolotareff reduction, lattice decoding algorithm, shortest vector, Voronoi-relevant vectors.

I. Introduction

In lattice theory, a *generator matrix* $\boldsymbol{B}$ is any real matrix with linearly independent rows. Hence $n \geq d$, where n is the number of columns and d, the *dimension*, is the number of rows. The *lattice* generated by $\boldsymbol{B}$ is

$$\Lambda(\boldsymbol{B}) \triangleq \left\{ \boldsymbol{u}\boldsymbol{B} : \quad \boldsymbol{u} \in \mathbb{Z}^d \right\} \tag{1}$$

and the rows of $\boldsymbol{B}$ are called *basis vectors.* The *closest-point problem*, or *decoding* for short, is the problem of finding, for a given lattice Λ and an input vector $\boldsymbol{x} \in \mathbb{R}^n$, a vector $\hat{\boldsymbol{x}} \in \Lambda$ such that $\|\boldsymbol{x} - \hat{\boldsymbol{x}}\| \leq \|\boldsymbol{x} - \boldsymbol{c}\|$ for all $\boldsymbol{c} \in \Lambda$. (In source coding, the closest-point problem is called *encoding,* see below.) Throughout this report, $\|\boldsymbol{z}\|$ denotes the Euclidean norm of $\boldsymbol{z}$.

The *Voronoi region* of a lattice point is the set of all vectors that can be decoded as this point, namely

$$\Omega(\Lambda, \boldsymbol{c}) \triangleq \{\boldsymbol{x} \in \mathbb{R}^n : \quad \|\boldsymbol{x} - \boldsymbol{c}\| \leq \|\boldsymbol{x} - \boldsymbol{c}'\|, \forall \boldsymbol{c}' \in \Lambda\} \tag{2}$$

where $\boldsymbol{c} \in \Lambda$. The *Voronoi diagram* of a lattice is the set of all its Voronoi regions. It is known that all Voronoi regions $\Omega(\Lambda, \boldsymbol{c})$ are convex polytopes, they are symmetrical with respect to reflection in $\boldsymbol{c}$, and they are translations of $\Omega(\Lambda, \boldsymbol{0})$, where $\boldsymbol{0}$ is the origin. Hence, for most purposes it is sufficient to study $\Omega(\Lambda, \boldsymbol{0})$.

In communication theory, lattices have been proposed both for use in modulation and in quantization. If a lattice is used as a code for the Gaussian channel, maximum likelihood decoding in the demodulator is a closest-point problem. Analogously, if a lattice is used as a codebook for vector quantization and the mean square error criterion is used, then the encoding of each input vector is equivalent to a closest-point search. Another instance of the closest-point problem can appear in the source decoder or in the modulator, if the lattice is truncated into a so-called Voronoi code [10]. Typical for these applications in communications is that the same lattice is employed for many input vectors.

Other applications where the closest-point problem arises include lattice design [2] and Monte Carlo second moment estimation [11]. In both cases, random vectors are generated uniformly inside a Voronoi region using a closest-point algorithm.

The closely related shortest-vector problem has been used in assessing the quality of random number generators [25, pp. 89–113], and reduction has an important application in cryptography [36]. These search problems, and the determination of some further lattice parameters through similar methods, will be discussed in Section VI.

The choice of method for solving the closest-point problem depends on the structure of the lattice. Intuitively, the more structure a lattice has, the faster can the closest point be found. For most of the classical lattices, very efficient search methods have been tailored [12, Ch. 20]. A more general approach is to represent the lattice with a trellis and use a trellis decoding algorithm such as the Viterbi algorithm [7, 17]. Finite-state trellises exist if and only if the lattice contains d mutually orthogonal vectors [38].

In this report we address the problem of finding the closest point in a general lattice. We assume that it possesses no exploitable structure. One example of where this problem arises is when a generator matrix is continuously adjusted, such as in numerical lattice design [2]. Another example is MS-optimal separation of the linear phase component from an arbitrary phase vector [15], which can be useful in, e.g., speech coding.

The complexity of the general closest-point problem as a function of d was analyzed by van Emde Boas in 1981, who showed that the problem is NP-hard [39]. Hence, all known algorithms for solving the problem optimally have exponential complexity. It is also NP-hard to find an approximate solution such that the ratio between the found distance and the optimal one is upper-bounded by a constant [5].

A common approach to the general closest-point problem is to identify a certain region in $\mathbb{R}^n$ within which the optimal lattice point must lie, then investigate all points in this region, possibly reducing its size dynamically. The earliest work in the field was done for the shortest-vector problem (see Section VI-A) in the context of assessing the quality of certain random number generators. The finite region to be searched in these algorithms is a parallelepiped with its axes parallel to the basis vectors [13], [14], [25, pp. 89–101, 110].

The development of closest-point algorithms follows two main branches, inspired by two seminal papers. Pohst in 1981 examined points inside a hypersphere [32], whereas Kannan in 1983 used a rectangular parallelepiped [22]. Both papers later appeared in revised and extended versions, Pohst's as [16] and Kannan's, following a paper by Helfrich [20], as [23].

The two strategies will be discussed at greater length in Section III-A.

A crucial parameter for the performance of these algorithms is the initial size of the region. Some suggestions to this point were given in [40] and [31] for the Pohst strategy and in [8] for the Kannan strategy. The latter paper also includes an extensive complexity analysis. Applications are discussed in [9, 31, 40, 42].

Another, more subtle, difference between the two strategies is implicit from the presentations of them. Grossly generalizing, the Pohst method is intended as a practical tool and the method by Kannan as a theoretical tool. Papers dealing with the Pohst strategy typically discuss issues of implementation and applications, whereas the Kannan-type papers focus on asymptotic complexity. This is probably the reason why the two strategies, despite having so much in common, have never been compared and evaluated against each other.

In [35], Schnorr and Euchner suggest an important improvement of the Pohst strategy, by examining the points inside the aforementioned hypersphere in a different order. In Sections V and VII-C, the strategies by Pohst, Kannan, and Schnorr-Euchner are compared with respect to computational complexity, and it is shown that the Schnorr-Euchner strategy is faster than the other two. A practical implementation of the Schnorr-Euchner strategy is presented in Section III-B.

II. Preliminaries

In the following, we will say that two lattices are *identical* if all lattice points are the same. Two generator matrices $\boldsymbol{B}_1$ and $\boldsymbol{B}_2$ generate identical lattices $\Lambda(\boldsymbol{B}_1) = \Lambda(\boldsymbol{B}_2)$ if and only if

$$\boldsymbol{B}_1 = \boldsymbol{W}\boldsymbol{B}_2 \tag{3}$$

where $\boldsymbol{W}$ is a square matrix with integer entries, whose determinant is 1 or -1.

A generator matrix $\boldsymbol{B}_2$ is a *rotated and reflected* representation of another generator matrix $\boldsymbol{B}_1$ if

$$\boldsymbol{B}_1 = \boldsymbol{B}_2\boldsymbol{Q} \tag{4}$$

where $\boldsymbol{Q}\boldsymbol{Q}^T = \boldsymbol{I}$. This can be regarded as a change of coordinate system. If $\boldsymbol{B}_2$ is square and lower triangular, we say that it is a *lower-triangular representation* of $\boldsymbol{B}_1$. Any

generator matrix has a lower-triangular representation $\boldsymbol{B}_2$, which is unique except that any set of columns of $\boldsymbol{B}_2$ can be negated. How to find a lower-triangular representation of any generator matrix is discussed in Section IV.

Two lattices are *equivalent* if they are congruent, that is, if one can be obtained from the other through scaling, rotation, and reflection. Two generator matrices $\boldsymbol{B}_1$ and $\boldsymbol{B}_2$ generate equivalent lattices if and only if

$$\boldsymbol{B}_1 = c\boldsymbol{W}\boldsymbol{B}_2\boldsymbol{Q} \tag{5}$$

where c is a real nonzero constant and $\boldsymbol{W}$ and $\boldsymbol{Q}$ obey the same conditions as for (3) and (4). The equivalence relation is denoted $\Lambda(\boldsymbol{B}_2) \cong \Lambda(\boldsymbol{B}_1)$.

The process of selecting a good basis for a given lattice, given some criterion, is called *reduction*. In many applications, it is advantageous if the basis vectors are as short as possible, and "reasonably" orthogonal to each other. For lattice search problems, this was first noted by Coveyou and MacPherson [13]. Two kinds of reduction will be discussed in the following.

KZ reduction is named after Korkine and Zolotareff [26], who defined this reduction criterion in 1873. To determine if a generator matrix represents a KZ reduced basis, it is convenient to study its lower-triangular representation. A lower-triangular square generator matrix

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \\ \vdots \\ \boldsymbol{b}_d \end{bmatrix} = \begin{bmatrix} b_{11} & 0 & \cdots & 0 \\ b_{21} & b_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ b_{d1} & b_{d2} & \cdots & b_{dd} \end{bmatrix} \tag{6}$$

is defined recursively to be KZ reduced if $d = 1$ or else the following hold:

- $\boldsymbol{b}_1$ is a shortest nonzero vector in $\Lambda(\boldsymbol{B})$,
- $|b_{i1}| \leq |b_{11}|/2$ for $i = 2, \ldots, d$,
- The submatrix

$$\begin{bmatrix} b_{22} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ b_{d2} & \cdots & b_{dd} \end{bmatrix} \tag{7}$$

is KZ reduced.

An arbitrary generator matrix $\boldsymbol{B}$ is KZ reduced if and only if its lower-triangular representation is KZ reduced. Every lattice has at least one KZ reduced generator matrix [33].

For situations when KZ reduction would be too time-consuming, *LLL reduction*, named after Lenstra, Lenstra, and Lovász, has been suggested [27]. A lower-triangular generator matrix (6) is LLL reduced if either $d = 1$ or else the following hold:

- $\|\boldsymbol{b}_1\| \leq (2/\sqrt{3})\|\boldsymbol{b}_2\|$,
- $|b_{i1}| \leq |b_{11}|/2$ for $i = 2, \ldots, d$,
- The submatrix (7) is LLL reduced.

As before, an arbitrary generator matrix is LLL reduced if its lower-triangular representation is LLL reduced.

Any KZ reduced matrix is clearly also LLL reduced. The motivation for the latter criterion is that there exists a more efficient algorithm to convert any $d \times n$ generator matrix into an LLL reduced one [27]. The algorithm, which operates in polynomial time in d and n, has become very popular in applications.

III. Closest-Point Search Algorithms

A. Conceptual Description

To understand lattice search algorithms, a recursive characterization of lattices is useful. Let the $d \times n$ matrix $\boldsymbol{B}$ be decomposed as

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}_1 \\ \boldsymbol{b}_d \end{bmatrix} \tag{8}$$

and let $\boldsymbol{b}_d = \boldsymbol{b}_{\|} + \boldsymbol{b}_{\perp}$, where $\boldsymbol{b}_{\|}$ is in the row space of $\boldsymbol{B}_1$ (the top $d-1$ rows of $\boldsymbol{B}$) and $\boldsymbol{b}_{\perp}$ is in the null space. If $\boldsymbol{B}$ is lower triangular as in (6), this decomposition is particularly simple, namely, $\boldsymbol{b}_{\|} = [b_{d1}, \ldots, b_{d,d-1}, 0]$ and $\boldsymbol{b}_{\perp} = [0, \ldots, 0, b_{dd}]$.

With the given terminology, any d-dimensional lattice can be decomposed as

$$\Lambda(\boldsymbol{B}) = \bigcup_{u_d=-\infty}^{\infty} \left\{ \boldsymbol{c} + u_d \boldsymbol{b}_{\|} + u_d \boldsymbol{b}_{\perp} : \quad \boldsymbol{c} \in \Lambda\left(\boldsymbol{B}_1\right) \right\} \tag{9}$$

which is basically a stack of $(d-1)$-dimensional translated sublattices. The $(d-1)$-dimensional hyperplanes that contain these sublattices will be called ($(d-1)$-dimensional) *layers* and the index u_d denotes which layer a certain lattice point belongs to. The vector $\boldsymbol{b}_{\|}$ is the offset which one sublattice is translated within its layer, with respect to an adjacent sublattice, and $\boldsymbol{b}_{\perp}$ is a normal vector to the layers, whose length equals the distance between two adjacent layers. This distance equals b_{dd} for lower-triangular generator matrices. Recalling that any generator matrix can be rotated into lower-triangular form, we will in this report let b_{ii} denote the distance between $(i-1)$-dimensional layers, even when no explicit triangular constraint is imposed.

Now the search algorithm for a d-dimensional lattice will be recursively described as a finite number of $(d-1)$-dimensional search operations. Let $\boldsymbol{x}$ be a vector to be decoded in the lattice $\Lambda(\boldsymbol{B})$, which is decomposed into layers according to (9). The orthogonal distance from $\boldsymbol{x}$ to the layer with index u_d is

$$y_d \triangleq |u_d - \tilde{u}_d| \cdot \|\boldsymbol{b}_{\perp}\| \tag{10}$$

where

$$\tilde{u}_d \triangleq \frac{\boldsymbol{x}\boldsymbol{b}_{\perp}^T}{\|\boldsymbol{b}_{\perp}\|^2}. \tag{11}$$

Suppose that an upper bound R_d is known on the attainable distance $\|\hat{\boldsymbol{x}} - \boldsymbol{x}\|$, where $\hat{\boldsymbol{x}}$ is a closest lattice point. Then it suffices to consider a finite number of the layers in (9) in order to ensure that the closest lattice point will be found. The indices of these layers are[1]

$$u_d = \left\lceil \tilde{u}_d - \frac{R_d}{\|\boldsymbol{b}_{\perp}\|} \right\rceil, \ldots, \left\lfloor \tilde{u}_d + \frac{R_d}{\|\boldsymbol{b}_{\perp}\|} \right\rfloor \tag{12}$$

since layers for which $y_d > R_d$ are not relevant. Of the considered layers, the one with $u_d = [\tilde{u}_d]$ has the shortest orthogonal distance to $\boldsymbol{x}$.

Four types of search methods will now be identified. They each search the layers indexed in (12), but they differ in the order in which the layers are examined and in the choice of upper bound R_{d-1} to be used in the $(d-1)$-dimensional search problems.

[1] The functions $\lfloor z \rfloor$, $\lceil z \rceil$, and $[z]$ denote the maximum integer not greater than z, the minimum integer not less that z, and the closest integer to z (ties are broken arbitrarily), respectively. In addition, $SGN(z)$ is -1 if $z \leq 0$ and 1 if $z > 0$. (The algorithm in Section III-B requires that ± 1 is returned even for $z = 0$, which may deviate from the operation of most built-in sign functions.)

If only $u_d = [\tilde{u}_d]$ is considered, the d-dimensional search problem is reduced to just one $(d-1)$-dimensional problem and no bound R_d is needed. Recursive application of this strategy yields *Babai's nearest plane algorithm* [6], which is a fast method (polynomial in the number of rows d and columns n of $\boldsymbol{B}$) to find a nearby lattice point. In general, the returned point ("Babai's point") is not the optimal one, but the error can be bounded.

The other three methods all find the optimal point. Scanning all layers in (12), and supplying each $(d-1)$-dimensional search problem with the same value of R_{d-1} regardless of u_d, yields the *Kannan strategy.* Variants of this strategy differ mainly in how the bounds R_k, $k = 1, \ldots, d$, are chosen [8, 20, 22, 23]. Geometrically, the Kannan strategy amounts to generating and examining all lattice points within a given rectangular parallelepiped.

The d-dimensional decoding error vector $\hat{\boldsymbol{x}} - \boldsymbol{x}$ consists, in the given recursive framework, of two orthogonal components, one in the row space of $\boldsymbol{B}_1$ and one parallel to $\boldsymbol{b}_\perp$. The former is the $(d-1)$-dimensional decoding error and the length of the latter is y_d, which varies with u_d. Hence R_{d-1} can safely be chosen as

$$R_{d-1} = \sqrt{R_d^2 - y_d^2}. \tag{13}$$

This idea of letting R_{d-1} depend on u_d is the *Pohst strategy* [16,31,32,40,42]. In geometrical terms, points inside a hypersphere, not a parallelepiped, are investigated. When any lattice point $\boldsymbol{x}'$ inside the sphere has been found, the bound R_d can be immediately updated to $\|\boldsymbol{x}' - \boldsymbol{x}\|$, since this is an obvious upper bound on $\|\hat{\boldsymbol{x}} - \boldsymbol{x}\|$.

The *Schnorr-Euchner strategy* [35] combines the advantages of Babai's nearest plane algorithm and the Pohst strategy. Assume that $\tilde{u}_d \leq [\tilde{u}_d]$. Then the sequence

$$u_d = [\tilde{u}_d], [\tilde{u}_d] - 1, [\tilde{u}_d] + 1, [\tilde{u}_d] - 2, \ldots \tag{14}$$

orders the layers according to nondecreasing distance from $\boldsymbol{x}$. (A trivial counterpart holds when $\tilde{u}_d > [\tilde{u}_d]$.) The advantages of examining the layers in this order are subtle but significant. Since the likelihood that a layer will contain $\hat{\boldsymbol{x}}$ decreases with increasing y_d (see (13), the chances of finding $\hat{\boldsymbol{x}}$ early is maximized. Another advantage with the nondecreasing distance y_d is that the search can safely be terminated as soon as $y_d > R_d$, where R_d is the distance to the best found lattice point so far. The very first lattice point generated will by definition be Babai's point. Since the ordering in (14) does not

depend on R_d, no initial bound on R_d is needed. A closest-point algorithm, based on the Schnorr-Euchner strategy, is further detailed in Sections III-B and IV.

B. Detailed Description

This subsection contains a stand-alone presentation of an efficient closest-point algorithm, based on the Schnorr-Euchner strategy. It is intended to be sufficiently detailed to allow a straightforward implementation, even with no study of the underlying theory. For efficiency, the recursive operations discussed in the previous subsection have been restructured into a loop. The variables $\boldsymbol{S}$ and $\hat{\boldsymbol{u}}$ are used instead of the more natural $\boldsymbol{B} = \boldsymbol{S}^{-1}$ and $\hat{\boldsymbol{x}} = \hat{\boldsymbol{u}}\boldsymbol{B}$ as input and output parameters. As discussed in the next subsection, this is motivated by the typical communication application in which many input vectors are decoded in the same lattice.

Some notation needs to be defined. Matrix and vector elements are named according to the following conventions:

$$\boldsymbol{u} = \begin{bmatrix} u_1 & \cdots & u_d \end{bmatrix}$$

$$\boldsymbol{p}_k = \begin{bmatrix} p_{k1} & \cdots & p_{kk} \end{bmatrix}, \quad k = 1, \ldots, d$$

$$\boldsymbol{S} = \begin{bmatrix} s_{11} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ & & \ddots & 0 \\ s_{d1} & & \cdots & s_{dd} \end{bmatrix}.$$

The integer operations $[z]$ and $SGN(z)$ are defined in footnote 1.

Input:

$\boldsymbol{S}$: A $d \times d$ lower-triangular matrix with positive diagonal elements.

$\boldsymbol{x}$: A d-dimensional vector to decode in $\Lambda(\boldsymbol{S}^{-1})$.

Output:

$\hat{\boldsymbol{u}}$: An integer vector such that $\hat{\boldsymbol{u}}\boldsymbol{S}^{-1}$ is a lattice point closest to $\boldsymbol{x}$.

Algorithm $Decode(\boldsymbol{S}, \boldsymbol{x})$**:**

$d :=$ the size of $\boldsymbol{S}$ $\quad *$ Dimension
$bestdist := \infty$ $\quad *$ Current distance record
$k := d$ $\quad *$ Dimension of examined layer
$dist_k := 0$ $\quad *$ Distance to examined layer
$\boldsymbol{p}_k := \boldsymbol{x}\boldsymbol{S}$ $\quad *$ Used to compute $\tilde{u}_d$, see (11)
$u_k := \lceil p_{kk} \rceil$ $\quad *$ Examined lattice point
$y := \dfrac{p_{kk} - u_k}{s_{kk}}$ $\quad *$ See (10)
$step_k := SGN(y)$ $\quad *$ Offset to next layer in (14)
Loop:
  $newdist := dist_k + y^2$
  If $newdist < bestdist$ **then {**
    If $k \neq 1$ **then {** *Case A*
      $p_{k-1,i} := p_{ki} - y s_{ki}, \quad i = 1, \ldots, k-1$
      $k := k - 1$ $\quad *$ Move down
      $dist_k := newdist$
      $u_k := \lceil p_{kk} \rceil$ $\quad *$ Closest layer
      $y := \dfrac{p_{kk} - u_k}{s_{kk}}$
      $step_k := SGN(y)$
    } else { *Case B*
      $\hat{\boldsymbol{u}} := \boldsymbol{u}$ $\quad *$ Best lattice point so far
      $bestdist := newdist$ $\quad *$ Update record
      $k := k + 1$ $\quad *$ Move up
      $u_k := u_k + step_k$ $\quad *$ Next layer
      $y := \dfrac{p_{kk} - u_k}{s_{kk}}$
      $step_k := -step_k - SGN\,(step_k)$
    }
  } else { *Case C*
    If $k = d$ **then Exit** and return $\hat{\boldsymbol{u}}$
    else {
      $k := k + 1$ $\quad *$ Move up
      $u_k := u_k + step_k$ $\quad *$ Next layer
      $y := \dfrac{p_{kk} - u_k}{s_{kk}}$
      $step_k := -step_k - SGN\,(step_k)$
    }
  }
Goto Loop

In this algorithm, k is the dimension of the sublayer structure currently being investigated. Each time a k-dimensional layer has been found to which the distance is shorter than the currently smallest found distance, this layer is expanded into $(k-1)$-dimensional sublayers. This is done in Case A. Conversely, as soon as the distance to an examined layer is greater than the lowest distance, the algorithm moves up one step in the hierarchy of layers, which is done in Case C. Case B is invoked when the algorithm has successfully moved down all the way to a 0-dimensional layer, that is, a lattice point, without superseding the lowest distance. Then this lattice point is stored as a potential output point, the lowest distance is updated, and the algorithm moves back up again, without restarting.

IV. Pre- and Postprocessing

The algorithm *Decode* presented in Section III-B requires that the lattice be represented by a lower-triangular generator matrix, whose diagonal elements are all positive. Such a representation can be found for any lattice (see (4)), so this requirement does not impose any constraint on the set of lattices that can be searched. Moreover, a representation with the required properties can be found in infinitely many ways for any given lattice, which leaves the user with the freedom of choosing one of them. The algorithm computes a closest vector regardless of the representation choice, but the speed with which it reaches the result varies considerably between different representations. This is the topic of this subsection: How should a given search problem be preprocessed, in order to make the most efficient use of *Decode*?

To address this question, we first present a general lattice search algorithm. It can be regarded as a "front end" of *Decode*, where explicit pre- and postprocessing is included to allow generator matrices that are not lower triangular, possibly not even square. As with *Decode*, we first describe the algorithm conceptually, then suggest how to implement it.

Assume that a generator matrix $\boldsymbol{B}$ and an input vector $\boldsymbol{x}$ are given. By linear integer row operations, we change $\boldsymbol{B}$ into another matrix, say $\boldsymbol{B}_2$, which represents an identical lattice. (The purpose is to speed up *Decode*, see below.) Next we rotate and reflect the lattice into a lower-triangular form, $\boldsymbol{B}_3$, so that $\Lambda(\boldsymbol{B}_3) \cong \Lambda(\boldsymbol{B}_2) = \Lambda(\boldsymbol{B})$. It is essential

to rotate and reflect the input vector $\boldsymbol{x}$ in a similar fashion, so that the transformed input vector, $\boldsymbol{x}_3$, has the same relation to $\Lambda(\boldsymbol{B}_3)$ as $\boldsymbol{x}$ has to $\Lambda(\boldsymbol{B})$. This can be regarded as a change of coordinate system. Now the search problem has a form that is suitable for *Decode*, which will find the closest lattice vector $\hat{\boldsymbol{x}}_3$ in this coordinate system. Reversing the operations of rotation and reflection produces $\hat{\boldsymbol{x}}$, the lattice vector closest to $\boldsymbol{x}$ in $\Lambda(\boldsymbol{B})$.

Following these steps, the algorithm is detailed as follows.

Input:

$\boldsymbol{B}$: A d-row, n-column generator matrix.

$\boldsymbol{x}$: An n-element vector to decode.

Output:

$\hat{\boldsymbol{x}}$: The lattice point closest to $\boldsymbol{x}$.

Algorithm $ClosestPoint(\boldsymbol{B}, \boldsymbol{x})$**:**

1. Let $\boldsymbol{B}_2 := \boldsymbol{W}\boldsymbol{B}$, where $\boldsymbol{W}$ is an $n \times n$ integer matrix with determinant ± 1.
2. Find an orthonormal matrix $\boldsymbol{Q}$ such that $\boldsymbol{B}_2 = \boldsymbol{B}_3\boldsymbol{Q}$, where $\boldsymbol{B}_3$ is a $d \times d$ lower-triangular matrix with positive diagonal elements.
3. Let $\boldsymbol{S}_3 := \boldsymbol{B}_3^{-1}$.
4. Let $\boldsymbol{x}_3 := \boldsymbol{x}\boldsymbol{Q}^T$.
5. Let $\hat{\boldsymbol{u}}_3 := Decode\,(\boldsymbol{S}_3, \boldsymbol{x}_3)$.
6. Return $\hat{\boldsymbol{x}} := \hat{\boldsymbol{u}}_3\boldsymbol{B}_2$.

Step 1, which is reduction, is optional. It is possible to select $\boldsymbol{W}$ as the identity matrix, which amounts to no reduction at all. This works well for low-dimensional and not too ill-conditioned generator matrices, as will be shown in Section VII. However, the speed and numerical stability of the search can be improved significantly by appropriate reduction, which is the topic of the last part of this subsection.

Step 2 implies rotation and reflection of the lattice, as in (4). The standard method to achieve this is QR factorization of $\boldsymbol{B}_2^T$, which gives both $\boldsymbol{Q}$ and $\boldsymbol{B}_3$ [19, pp. 208–236], [37, pp. 166–176]. ($\boldsymbol{B}_3$ is equal to $\boldsymbol{R}^T$ in the QR factorization.) QR factorization can be understood as a change of coordinate system: Measure the first coordinate along $\boldsymbol{b}_1$, the second in the plane spanned by $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$, etc. The generator matrix will in this new coordinate system be square and lower triangular.

For *Decode* to work, care must be taken to ensure that all diagonal elements of $\boldsymbol{B}_3$ are positive. Some implementations of QR factorization do not do this automatically; if this is the case, we multiply all columns of $\boldsymbol{B}_3$ that contain a negative diagonal element, and the corresponding rows of $\boldsymbol{Q}$, by -1. As an alternative to QR factorization, $\boldsymbol{B}_3$ can be obtained by Cholesky factorization of $\boldsymbol{B}_2\boldsymbol{B}_2^T$ [19, pp. 84–93], [37, pp. 332–334], after which the rotation matrix is given by $\boldsymbol{Q} = \boldsymbol{B}_3^{-1}\boldsymbol{B}_2$.

In Steps 4–6, the input vectors are processed. They are transformed into the coordinate system of $\boldsymbol{B}_3$, decoded, and transformed back again.

If a large set of vectors are to be decoded for the same lattice, Steps 1–3 are of course only carried out once for the whole set. In this case, the overall execution time may benefit from an effective but time-consuming reduction method being applied in Step 1. To understand precisely what kind of preprocessing would improve the performance of the search algorithm, we recall the recursive interpretation of lattices and of the algorithm from Section III-A. A d-dimensional lattice consists of parallel $(d-1)$-dimensional sublattices, translated and stacked on top of each other. This decomposition into sublattices is controlled by the reduction method. Two properties of the decomposition are desirable for a given lattice:

(a) The $(d-1)$-dimensional layers should be as far apart as possible. This minimizes the number of layers to investigate, as all layers within a certain distance range need to be scanned. As an extreme case, suppose that the spacing between $(d-1)$-dimensional layers is much larger than any other k-dimensional layer spacing in the lattice. Then the closest point will always lie in the closest $(d-1)$-dimensional layer and the dimensionality of the problem is practically reduced by one.

(b) The 0-dimensional layers (lattice points) should be as densely spaced as possible in the 1-dimensional layers (lines). The denser they are, the higher is the probability that the closest lattice point will belong to the closest lattice line. If the 1-dimensional spacing is much smaller than all the other inter-layer distances, the closest point will always lie in the closest line, so the dimensionality of the problem is practically reduced by one.

Both observations can of course be applied recursively, and hence high-dimensional layer spacing should be large, and low-dimensional spacing should be small. This suggests

two greedy algorithms: (a) sequentially maximizing the distances between k-dimensional layers, beginning at $k = d-1$, and (b) minimizing the same distances, beginning at $k = 0$.

These two goals are each other's duals in a fairly strict sense. Even though they may appear contradictory, they are in fact very similar [25, p. 94–98]. A reduction algorithm can choose the numbers $\{b_{kk}\}$ in many ways for a given lattice, but their product is invariant; it equals the volume of a Voronoi region. Now (a) is solved by maximizing first b_{dd}, then $b_{d-1,d-1}$, etc. Because of the constant product, this procedure forces low values into the last elements b_{11}, b_{22}, etc., so a good solution of (a) is in general good for (b) too. Conversely, (b) is solved by first minimizing b_{11}, then b_{22}, etc., which automatically yields a good basis in sense (a), too.

The smallest possible value of b_{11} that can be selected for a given lattice equals the length of the shortest vector in the lattice.[2] Also, the largest possible b_{dd} is the reciprocal of the length of the shortest vector in the dual lattice.[3] Applying these shortest-vector criteria recursively, we conclude that (b) is solved optimally by KZ reduction of any basis for the lattice. This follows directly from the recursive definition of KZ reduction in Section II. Also, (a) is solved optimally by KZ reduction of a basis for the dual lattice, then reversing the order of the rows, and finally transposing the inverse of the resulting matrix. (In the following, we will refer to this latter strategy as "KZ reduction of the dual".) Finally, the LLL algorithm yields an approximate (but faster) solution to both (a) and (b), because of its inherent sorting mechanism.

Our recommendation, which is supported in Section VII, is to use KZ reduction in applications where the same lattice is to be searched many times, otherwise LLL.

V. Complexity Analysis

Banihashemi and Khandani observed that the average complexity of a search method for uniformly distributed input vectors[4] is proportional to the volume of the region being searched [8]. They used this volume to assess the complexity of the Kannan algorithm. We adopt the same approach here to analyze *ClosestPoint* (which is based on the Schnorr-

[2] Shortest-vector problems can be solved by a variant of *ClosestPoint*, as detailed in Section VI-A.

[3] Because a generator matrix for the dual lattice is $(\boldsymbol{B}^{-1})^T$, provided that $\boldsymbol{B}$ is square.

[4] In the context of lattices, a "uniform distribution" is assumed to be uniform over a region large enough to make boundary effects negligible. This is equivalent to having a uniform distribution over just one Voronoi region.

Euchner strategy), and to compare it to Kannan. A comparison between *ClosestPoint* and an algorithm based on the Pohst strategy is carried out experimentally, in Section VII.

For a given lattice, let $V_k(R)$ denote the volume searched in a k-dimensional layer, when R is the given upper bound on the attainable distance. Since the algorithm does not require an initial value for R_d, the desired complexity measure is $V_d(\infty)$.

Theorem 1:

$$\text{(a)} \quad V_d(\infty) \leq \prod_{k=1}^{d} \beta_k \tag{15}$$

$$\text{(b)} \quad V_d(\infty) \leq \left(\frac{d}{2\pi e}\right)^{-d/2} \beta_d^d \tag{16}$$

where

$$\beta_k \triangleq \left(\sum_{i=1}^{k} b_{ii}^2\right)^{1/2}. \tag{17}$$

Proof: The algorithm always begins by comparing the currently best upper bound R_k with the distance between the input vector $\boldsymbol{x}$ and Babai's point.[5] The distance to Babai's point in k dimensions is, for any $\boldsymbol{x}$, at most $\beta_k/2$ [6]. Denote the smaller of the two distances by $f_k(R_k) \triangleq \min(R_k, \beta_k/2)$. Also, let y_k denote the orthogonal distance from $\boldsymbol{x}$ to a $(k-1)$-dimensional layer to be scanned. The algorithm considers all layers for which $y_k \leq f_k(R_k)$, and the distance upper bound imposed on each of these layers is

$$R_{k-1} = \sqrt{f_k^2(R_k) - y_k^2}. \tag{18}$$

The volume $V_k(R_k)$, regarded as an integral over $V_{k-1}(R_{k-1})$, is hence recursively bounded by

$$V_k(R) \leq 2 \int_0^{f_k(R)} V_{k-1}(\sqrt{f_k^2(R) - y^2}) dy, \quad \text{if } k \geq 1 \tag{19}$$

$$V_0(R) = 1 \tag{20}$$

where the index of R has been dropped.

[5] In the implementation given in Section III-B, R_k corresponds to $(bestdist - dist_k)^{1/2}$.

In the absence of a closed form for this quantity, we define

$$V'_k(R) \triangleq 2\int_0^{\beta_k/2} V'_{k-1}(\sqrt{\beta_k^2/4 - y^2})dy, \qquad \text{if } k \geq 1 \tag{21}$$

$$V''_k(R) \triangleq 2\int_0^{R} V''_{k-1}(\sqrt{R^2 - y^2})dy, \qquad \text{if } k \geq 1 \tag{22}$$

$$V'_0(R) \triangleq V''_0(R) \triangleq 1 \tag{23}$$

and observe that $V_k(R) \leq V'_k(R)$ and $V_k(R) \leq V''_k(R)$. The recursions (21)–(23) are solved by, respectively

$$V'_k(R) = \prod_{j=1}^{k} \beta_j \tag{24}$$

$$V''_k(R) = \frac{2^k \pi^{k/2}}{\Gamma(k/2+1)} R^k \tag{25}$$

which is easily verified. Part (a) of the theorem is proved by (24). To complete the proof of (b), we observe that $V_k(R) = V_k(f_k(R))$ and consequently

$$V_d(\infty) = V_d(\beta_d/2) \tag{26}$$

$$\leq \frac{\pi^{d/2}}{\Gamma(d/2+1)} \beta_d^d \tag{27}$$

$$\leq \left(\frac{d}{2\pi e}\right)^{-d/2} \beta_d^d \tag{28}$$

where the last inequality follows from Stirling's inequality. □

The corresponding volume K_d for the Kannan algorithm is known exactly. For every lattice, it is in the range

$$\prod_{k=1}^{d} \beta_k \leq K_d \leq \beta_d^d \tag{29}$$

where the lower bound is exact if the sequence $b_{11}, \ldots, b_{dd}$ is increasing and the upper bound is exact if it is decreasing [8]. For a "good" lattice, this sequence generally displays a decreasing trend, but the decrease is not necessarily monotonic [24], [12, p. 158]. Hence, K_d is often close to the upper bound.

The *ClosestPoint* algorithm is faster than the Kannan algorithm for all dimensions and all lattices, since the upper bound (15) for *ClosestPoint* coincides with the lower bound for

Kannan (29). The magnitude of the gain is suggested by Theorem 1 (b): For lattices such that the upper bound in (29) is exact, *ClosestPoint* is at least a factor $(d/2\pi e)^{d/2}$ faster. This factor is meant to indicate the asymptotic relation, and smaller order components have been dropped in (28). For moderate values of d, (27) yields a significantly better bound, and the factor is always at least one, even for "bad" lattices.

Banihashemi and Khandani point out that the covering radii of the lattice and its sublattices can be exploited to reduce the complexity of the Kannan algorithm [8]. This option can be included in *ClosestPoint* as well. However, it is difficult to determine the covering radius of a general lattice. The only known algorithm is the "diamond-cutting algorithm" [41], which, as detailed in Section VI-C, is confined by memory limitations to low dimensions. Methods to upper-bound the covering radius can be used instead [40].

VI. More Lattice Search Problems

Other search problems involving lattices can be solved by modifications and extensions of the *ClosestPoint* algorithm. These include finding lattice parameters such as the shortest vector (or, equivalently, the packing density [12, p. 10]), the kissing number, and the Voronoi-relevant vectors. *ClosestPoint* can also be used to perform the key step in basis reduction.

A. Shortest Vector

Given a lattice Λ, the *shortest-vector problem* is to find the vector in $\Lambda - \{\mathbf{0}\}$ with the smallest Euclidean norm. Its history is closely interlinked with that of the closest-point problem. It has been conjectured that shortest-vector problem is NP-hard [39], but, in contrast to the closest-point problem, this has not been proved. The conjecture was supported by the result of Ajtai, who showed that the shortest vector problem for randomized reductions is NP-hard [4], and by Micciancio [28], who showed that to find an approximate solution within any constant factor less than $\sqrt{2}$ is also NP-hard for randomized reductions. It has also been proven that the shortest vector problem is not harder than the closest-vector problem [18, 21].

The *ClosestPoint* algorithm can be straightforwardly modified to solve the shortest-vector problem instead. The general idea is to submit $\boldsymbol{x} = \mathbf{0}$ as the input and disregard

$\hat{\boldsymbol{x}} = \mathbf{0}$ as a potential closest vector. Algorithmically, the changes needed to convert *ClosestPoint* into *ShortestVector* are the following.

- Omit $\boldsymbol{x}$ as an input for *Decode* and *ClosestPoint*.
- In *ClosestPoint*, skip Step 4.
- In *Decode*, replace line 5 with "$\boldsymbol{p}_k := \mathbf{0}$".
- Replace lines 1–2 of Case B with

If $newdist \neq 0$ **then {**

$\hat{\boldsymbol{u}} := \boldsymbol{u}$

$bestdist := newdist$

}.

In any lattice, there is an even number of shortest vectors, because the lattice is symmetrical with respect to reflection in $\mathbf{0}$. Hence if $\hat{\boldsymbol{x}}$ is a shortest vector, so is $-\hat{\boldsymbol{x}}$. If execution time is crucial, a factor of 2 in computation time can be gained by exploiting this symmetry. This is achieved by rewriting *Decode* to scan only half of the candidates $\boldsymbol{u}$, say, the ones for which the first nonzero component is positive.

B. Kissing Number of Lattices

The *kissing number* of a lattice Λ is defined as the number of shortest nonzero vectors in Λ. To compute this lattice parameter, it is essential to employ infinite precision; an arbitrarily small perturbation of a generator matrix has the potential of reducing the kissing number to two, regardless of the original value. However, we do not recommend implementing *Decode* using exact arithmetics. The same goal can be achieved far more efficiently by implementing the time-consuming operations as before using finite-precision real numbers, in conjunction with a final infinite-precision stage, where a finite set of candidates is evaluated.

The new version of *Decode* needs to keep track of a set of potential shortest vectors, not just the one best candidate. A margin of accuracy must be included in the comparisons, to avoid missing some of the shortest vectors due to numerical errors. This is achieved by the following changes, which convert *ShortestVector* into *KissingNumber*.

(i) In *Decode*, include "$\hat{\mathcal{U}} := \varnothing$" among the initial assignments.

(ii) In *Decode*, replace the line preceding Case A with "**If** $newdist < (1+\epsilon)bestdist$ **then {**", where ϵ is a small positive number.

(iii) Replace the first two assignments in Case B with "$\hat{\mathcal{U}} := \hat{\mathcal{U}} \cup \{\boldsymbol{u}\}$" and "$bestdist := \min(bestdist, newdist)$".

(iv) Replace $\hat{\boldsymbol{u}}$ in Case C with $\hat{\mathcal{U}}$, and replace $\hat{\boldsymbol{u}}_3$ in Step 5 with $\hat{\mathcal{U}}_3$.

(v) Remove "$k := k+1$" from Case B.

(vi) Replace Step 6 with

6. Compute exact values of $\|\boldsymbol{u}\boldsymbol{B}_2\|$ for all $\boldsymbol{u} \in \hat{\mathcal{U}}_3$ and return the number of occurrences of the lowest value.

As for the shortest-vector problem, a variant of the closest-point problem can be formulated that in case of a tie returns all the lattice points that have minimum distance to the input vector, not just one of them. *ClosestPoint* is converted into *AllClosestPoints* through the following modifications.

- Apply the changes (i)–(v) above to *ClosestPoint*.
- Replace Step 6 with

6. Compute exact values of $\|\boldsymbol{u}\boldsymbol{B}_2 - \boldsymbol{x}\|$ for all $\boldsymbol{u} \in \hat{\mathcal{U}}_3$ and call the lowest value $bestdist$. Return $\hat{\mathcal{X}} := \{\boldsymbol{u}\boldsymbol{B}_2 : \boldsymbol{u} \in \hat{\mathcal{U}}_3, \|\boldsymbol{u}\boldsymbol{B}_2 - \boldsymbol{x}\| = bestdist\}$.

The main application of this algorithm lies in the solution of the next problem.

C. Voronoi-Relevant Vectors

The *relevant-vector problem* is to find the facets[6] of the Voronoi region $\Omega(\Lambda, \mathbf{0})$, in other words, to find a minimal set $\mathcal{N}(\Lambda) \subseteq \Lambda$ for which

$$\Omega(\Lambda, \mathbf{0}) = \{\boldsymbol{x} \in \mathbb{R}^n : \quad \|\boldsymbol{x}\| \leq \|\boldsymbol{x} - \boldsymbol{c}'\|, \forall \boldsymbol{c}' \in \mathcal{N}(\Lambda)\}\,. \tag{30}$$

The vectors in $\mathcal{N}(\Lambda)$ are called *Voronoi-relevant*, or simply *relevant*. Our method to solve the problem is through the following theorem.

Theorem 2: The Voronoi regions of two lattice points $\boldsymbol{c}_1 \in \Lambda$ and $\boldsymbol{c}_2 \in \Lambda$ share a facet if and only if

$$\|\boldsymbol{m} - \boldsymbol{c}_1\| < \|\boldsymbol{m} - \boldsymbol{c}'\|, \forall \boldsymbol{c}' \in \Lambda - \{\boldsymbol{c}_1, \boldsymbol{c}_2\} \tag{31}$$

[6]A *facet* is a $(d-1)$-dimensional face of a d-dimensional polytope.

where

$$\boldsymbol{m} = \frac{1}{2}\left(\boldsymbol{c}_1 + \boldsymbol{c}_2\right). \tag{32}$$

Proof: From (31) and (32),

$$\boldsymbol{m} \in \Omega(\Lambda, \boldsymbol{c}_1) \cap \Omega(\Lambda, \boldsymbol{c}_2) - \bigcup_{\boldsymbol{c}' \in \Lambda - \{\boldsymbol{c}_1, \boldsymbol{c}_2\}} \Omega(\Lambda, \boldsymbol{c}') \tag{33}$$

which completes the "if" part of the theorem. To prove the "only if" part, assume that $\Omega(\Lambda, \boldsymbol{c}_1)$ and $\Omega(\Lambda, \boldsymbol{c}_2)$ have a common facet. Let $\boldsymbol{x}$ be any point in the interior of this facet, so that

$$\|\boldsymbol{x} - \boldsymbol{c}_1\| = \|\boldsymbol{x} - \boldsymbol{c}_2\| < \|\boldsymbol{x} - \boldsymbol{c}'\|, \forall \boldsymbol{c}' \in \Lambda - \{\boldsymbol{c}_1, \boldsymbol{c}_2\}. \tag{34}$$

Then for all $\boldsymbol{c}'' \in \Lambda - \{\boldsymbol{c}_1, \boldsymbol{c}_2\}$

$$\begin{aligned} \|\boldsymbol{m} - \boldsymbol{c}_1\|^2 - \|\boldsymbol{m} - \boldsymbol{c}''\|^2 &= \frac{1}{2}\left(\|\boldsymbol{x} - \boldsymbol{c}_1\|^2 - \|\boldsymbol{x} - \boldsymbol{c}''\|^2\right) \\ &\quad + \frac{1}{2}\left(\|\boldsymbol{x} - \boldsymbol{c}_2\|^2 - \|\boldsymbol{x} - (\boldsymbol{c}_1 + \boldsymbol{c}_2 - \boldsymbol{c}'')\|^2\right) \\ &< 0 \end{aligned} \tag{35}$$

where the inequality follows from applying (34) twice. This proves (31). □

This theorem was given by Voronoï in a slightly different context [43, vol. 134, pp. 277–278], [12, p. 475], based on theory by Minkowski [29, pp. 81–85], [30, pp. 120–121]. Similar properties have been proved for the Voronoi regions of binary linear block codes [1] and of parallelepipeds [3].

Any vector $\boldsymbol{m}$ given by (32) has the form $\boldsymbol{z}\boldsymbol{B}$, where $2\boldsymbol{z} \in \mathbb{Z}^d$. However, to determine $\mathcal{N}(\Lambda(\boldsymbol{B}))$ for a given lattice $\Lambda(\boldsymbol{B})$, it is sufficient to investigate (31) for vectors $\boldsymbol{m}$ in the finite set

$$\mathcal{M}(\boldsymbol{B}) \triangleq \left\{\boldsymbol{z}\boldsymbol{B}: \quad \boldsymbol{z} \in \{0, 1/2\}^d - \{\boldsymbol{0}\}\right\}. \tag{36}$$

Any feasible vector $\boldsymbol{m} = \boldsymbol{z}\boldsymbol{B} \notin \mathcal{M}(\boldsymbol{B})$ can be mapped into another vector $\boldsymbol{m}' \in \mathcal{M}(\boldsymbol{B})$ by translating the lattice, except when $\boldsymbol{z} \in \mathbb{Z}^d$. But in this case it is obvious from Theorem 2 that $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ do not share a facet. This observation leads to the following algorithm.

Input:

$\boldsymbol{B}$: A d-row, n-column generator matrix.

Output:

$\mathcal{N}$: The relevant vectors of Λ.

Algorithm $RelevantVectors(\boldsymbol{B})$**:**

1. Let $\mathcal{N} := \varnothing$.
2. For all $\boldsymbol{m} \in \mathcal{M}(\boldsymbol{B})$,
   (a) Let $\hat{\mathcal{X}} := AllClosestPoints(\boldsymbol{B}, \boldsymbol{m})$.
   (b) If $|\hat{\mathcal{X}}| = 2$, let $\mathcal{N} := \mathcal{N} \cup \{2(\hat{\boldsymbol{x}} - \boldsymbol{m}) : \hat{\boldsymbol{x}} \in \hat{\mathcal{X}}\}$.
3. Return $\mathcal{N}$.

Optional optimization includes moving Steps 1–3 of $AllClosestPoints$ out of the loop, since all calls to $AllClosestPoints$ concern the same lattice. There is also a factor of 2 in complexity to be gained through the same symmetry argument as for $ShortestVector$.

It is readily seen that the maximum number of facets that a Voronoi region can have in any d-dimensional lattice is $2|\mathcal{M}(\boldsymbol{B})| = 2^{d+1} - 2$, and that this number is attained with probability 1 by a lattice whose basis is chosen randomly from a continuous distribution. These properties were proved by Minkowski in 1897 [30, pp. 120–121] and by Voronoï in 1909 [43, vol. 134, pp. 198–211 and vol. 136, pp. 67–70], respectively.

Relevant vectors have been determined for many classical lattices [12, Chs. 4,21], but we believe that the algorithm $RelevantVectors$ proposed above is the fastest known in the general case. The only alternative algorithm known to the authors is the "diamond-cutting algorithm" by Viterbo and Biglieri, which finds the complete geometrical description of the Voronoi region of any lattice [41]. This description includes all vertices, edges, etc., which evidently includes information on the relevant vectors. However, to employ the diamond-cutting algorithm for the sole purpose of determining the relevant vectors is inefficient. Voronoï showed in his classical work [43] that the number of $(d-k)$-dimensional faces of a d-dimensional lattice Voronoi region is upper-bounded by

$$(k+1)\sum_{i=0}^{k}(-1)^i\binom{k}{i}(k-i+1)^d \tag{37}$$

and that there exist lattices whose Voronoi regions attain this number for every k. The lattice nowadays called A_d^* is one example [43, vol. 136, pp. 74–82, 137–143]. Evaluating

(37) for $k = d, d-1, \ldots$ shows that the maximum number of vertices equals $(d+1)!$, the maximum number of edges is $(d/2)(d+1)!$, etc., which implies that the memory requirements for the diamond-cutting algorithm grows very rapidly with the dimension. This property confines its use to low dimensions.

RelevantVectors, on the other hand, uses negligible memory but does not fully determine the Voronoi regions, only their facets. In cases where vertices, etc., are desired, we suggest preceding the diamond-cutting algorithm with *RelevantVectors*, since the complexity (both time and memory) of the diamond-cutting algorithm can be reduced by incorporating knowledge of the relevant vectors.

D. Reduction

The last problem we mention here is the *reduction problem*. The problem of finding a KZ reduced basis for a lattice has already been mentioned in Sections II and IV. Theoretical results are available for specific lattices [24]. Algorithms for general lattices have been proposed by Kannan [23] and Schnorr [34]. Since KZ reduction essentially consists of solving d shortest-vector problems, a closest-point algorithm can be used in this context, too. In our experiments (Section VII) we have computed KZ reduced bases with this method.

The general strategy is to find a shortest vector in the lattice, project the lattice onto a hyperplane orthogonal to this vector, and find a KZ reduced basis of this $(d-1)$-dimensional lattice by recursion. In this application of *ShortestVector*, Step 1 is performed using LLL reduction, since KZ reduction is obviously not a usable prerequisite for KZ reduction. The details of implementation, which we omit, follow straightforwardly from the definition in Section II.

VII. Experiments

In this section, we report on experiments with the *ClosestPoint* algorithm from Section III-B, to evaluate its performance (in terms of search time) for low- and high-dimensional problems, to compare it with other similar algorithms, and to find out how the basis for the lattice should be preprocessed in order to achieve the best performance.

A. Experiment Setup

To evaluate the performance of the *ClosestPoint* algorithm, we must decide what class of lattices to use. The closest-point search methods studied here are general methods, and they do not compete well with algorithms specially designed for searching a particular lattice; such algorithms can exploit structure in the lattice and are generally faster (see Section I). Therefore, the natural application is one where a special search method for the chosen lattice does not exist. Here, we concentrate our efforts on experiments with random lattices without any structure that can be exploited. However, for comparison, we also include some experiments where the algorithms were applied to classical, highly structured lattices, such as the Leech lattice in 24 dimensions, and the cubic lattice $\mathbb{Z}^d$.

Following the discussion above, we use generator matrices with random elements, drawn from i.i.d. zero-mean Gaussian distributions. For each point in the diagrams 50 random matrices are generated, and for each matrix a large number of uniform random vectors (see footnote 4) are drawn.[7] We average over both random matrices and random input vectors. The search times for all the algorithms are averaged using the same matrices and the same set of input vectors. The results are given as average search time (in seconds), using a computer based on a 300 MHz Pentium II processor.

B. Preprocessing

An important question for a closest-point algorithm is whether the performance can be improved by somehow preprocessing the generator matrix. Since the preprocessing step needs to be executed only once, and the processed basis is typically used many times in most communication applications, it is usually worth the effort to invoke a good preprocessing procedure. In Section IV, three different preprocessing strategies are discussed: LLL reduction, KZ reduction, and KZ reduction of the dual. All of these basically aim to find as short and orthogonal basis vectors as possible, and here we present experiments to find the best of the reduction methods.

In Figure 1, the results from simulations using the three reduction methods are given.[8]

[7] The exact number is dependent on the dimension; for large dimensions with long search times the average is computed over about 200 vectors for each of the 50 matrices, and for small dimensions the number of vectors is much larger.

[8] As discussed previously, the reduction is typically performed only once for a large set of vectors, and the time

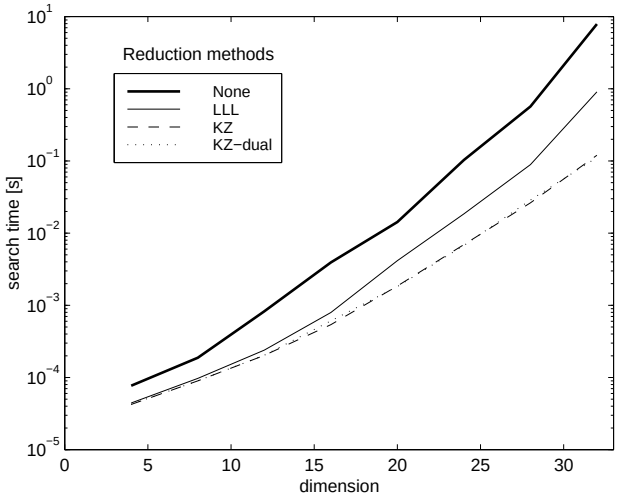

Fig. 1. Comparison between different reduction algorithms for preprocessing of the generator matrix. The average search time is plotted as a function of the dimension.

We see that the performance can be significantly improved by selecting a good preprocessor. The best methods in our study are the ones based on KZ reduction, which are almost indistinguishable. For high dimensions (30+), the KZ reduction can lower the search times by almost two orders of magnitude compared to unreduced bases, and by one order of magnitude compared to LLL reduction. However, up to about 10–15 dimensions, the LLL algorithm gives good results.

C. Algorithm Comparison

To assess the performance of the *ClosestPoint* algorithm, we have also implemented an algorithm described by Viterbo and Boutros [42], based on the Pohst strategy. The *ViterboBoutros* algorithm needs a starting bound on the attainable distance (see Section III-A). First we find the distance to Babai's point and then use it as an initial distance bound in the *ViterboBoutros* algorithm.

In Figure 2, the average time for a single closest-point operation is plotted as a function of the dimension, for the *ClosestPoint* and the *ViterboBoutros* algorithms.[9] The *ClosestPoint* algorithm is faster for all tested dimensions. We can also see that the speed

needed for reduction is not included in the search time results.

[9]For both algorithms, KZ reduction is applied to the generator matrices.

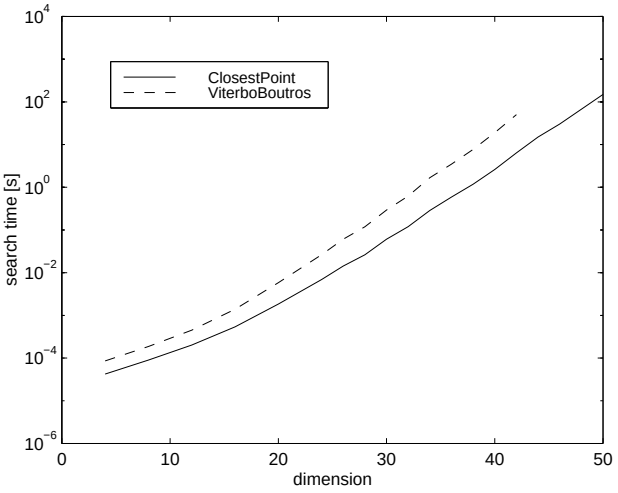

Fig. 2. Comparison of average search time for the *ClosestPoint* algorithm and the *ViterboBoutros* algorithm.

advantage increases with the dimension. In Figure 3, the speed ratio between the two algorithms is plotted as a function of dimension. For small dimensions, the speed advantage is a factor of 2, while for the 40-dimensional example, the speed ratio is about 8.

Why is the ClosestPoint algorithm faster? The main difference between the two algorithms is the order in which the layers are examined (see the discussion in Section III-A), but some implementation issues also differ, such as the use of a dual basis for internal representation, and a different loop structure. To determine which of the above differences leads to the speed difference, we reprogrammed *ClosestPoint* to examine the layers in the same order as in the *ViterboBoutros* algorithm, while still keeping the implementation structure the same. The results (not presented here) reveal that with the ordering of layers as in the *ViterboBoutros* algorithm, the *ClosestPoint* algorithm is no longer faster; the run times for both algorithms are similar. The conclusion is that it is the order in which the layers are examined that gives the *ClosestPoint* algorithm its better performance.

D. Comparison with Classical Lattices

To further illustrate the performance of *ClosestPoint*, we here evaluate its performance for classical lattices, and compare with the performance for random matrices (chosen from an i.i.d. Gaussian source). In Figure 4, the average search time for random lattices and for

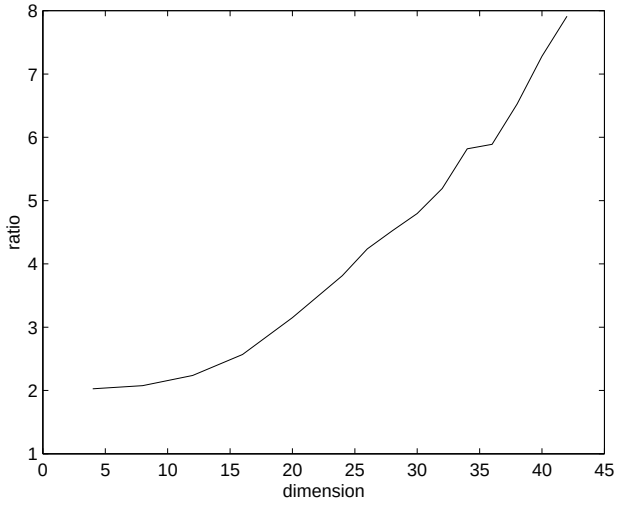

Fig. 3. The ratio between search times for $ClosestPoint$ and the $ViterboBoutros$ algorithm as a function of the dimension.

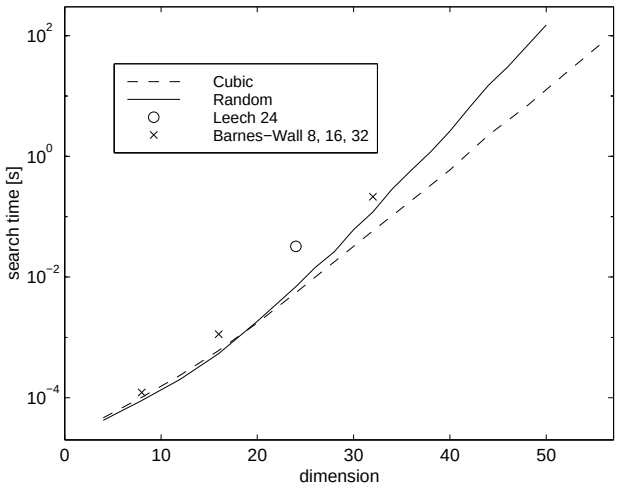

Fig. 4. Search time comparison of classical and random lattices.

the cubic lattice is plotted as a function of the dimension, together with the search times for the Leech lattice in 24 dimensions, and for the Barnes-Wall lattices in dimensions 8, 16, and 32. ¿From this figure, we see that there are no surprises in the search complexity for classical lattices; the general curve is the same as that for random lattices. This is the strength as well as the weakness of algorithms of this type; they do not rely on any particular structure. Also for the classical lattices (except the cubic lattice of course), KZ

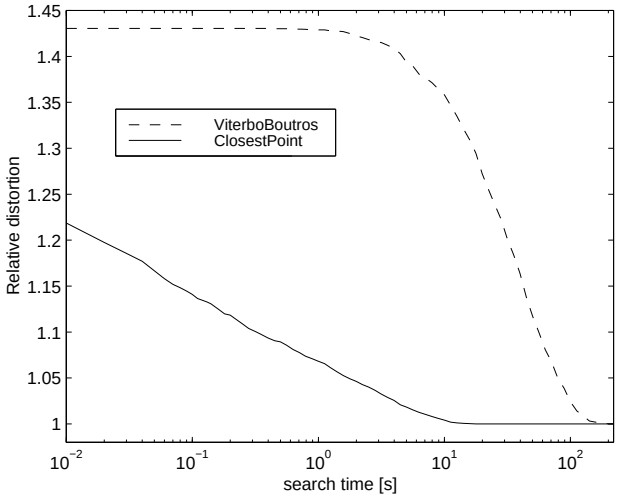

Fig. 5. Search time comparison for the *ClosestPoint* algorithm and the *ViterboBoutros* algorithm, for a 45-dimensional example.

reduction leads to faster search times, and is therefore applied before the experiments.

E. Suboptimal Search

The closest-point search algorithms studied here always returns a lattice point closest to the input point. However, in certain applications, it may be necessary to abort the search before the closest point has been found. Therefore, we have included experiments where the *ViterboBoutros* and the *ClosestPoint* algorithms are aborted after a given time. In Figure 5, the average ratio between the suboptimal and the optimal distortions is given for a 45-dimensional example, as a function of the time allowed for the search. From this figure, we see that the *ClosestPoint* algorithm quickly finds lattice points fairly close to the optimal one, while it takes a considerable amount of time until the *ViterboBoutros* algorithm improves on Babai's point. (Babai's point is the first point found for both algorithms; in this example, its average distortion is 1.43 times the distortion of the optimal point.)

We see that if a 10% higher average distortion than the optimal can be tolerated, the *ClosestPoint* algorithm is more than 150 times faster than the *ViterboBoutros* algorithm, as compared to about 10 times faster if we wait until the optimal point is found (see Figure 3). If 20% higher distortion can be tolerated for both algorithms, the *ClosestPoint*

algorithm is more than 1500 times faster.

We only report results for a single 45-dimensional example, but the general conclusion is the same for all tested dimensions; if we abort the search procedure before the optimal point is found, the speed advantage of the *ClosestPoint* algorithm over the *ViterboBoutros* algorithm is even clearer than for uninterrupted search.

VIII. Conclusion

Algorithms for finding the closest point in a lattice were studied. The complexity of different strategies for closest-point search was theoretically and experimentally evaluated, and a complete implementation of an efficient closest-point algorithm were given. The closest point algorithm were also extended to solve a number of related lattice search problems.

References

[1] E. Agrell, "On the Voronoi neighbor ratio for binary linear block codes," *IEEE Trans. Inform. Theory*, vol. 44, pp. 3064–3072, Nov. 1998.

[2] E. Agrell and T. Eriksson, "Optimization of lattices for quantization," *IEEE Trans. Inform. Theory*, vol. 44, pp. 1814–1828, Sept. 1998.

[3] E. Agrell and T. Ottosson, "ML optimal CDMA multiuser receiver," *Electronics Letters*, vol. 31, pp. 1554–1555, Aug. 1995.

[4] M. Ajtai, "The shortest vector problem in L_2 is NP-hard for randomized reductions," Rep. TR97-047, rev. 01, Electronic Colloquium on Computational Complexity, Univ. of Trier, Germany, Nov. 1997.

[5] S. Arora, L. Babai, J. Stern, and Z. Sweedyk, "The hardness of approximate optima in lattices, codes, and systems of linear equations," *Journal of Computer and System Sciences*, vol. 54, pp. 317–331, Apr. 1997.

[6] L. Babai, "On Lovász' lattice reduction and the nearest lattice point problem," *Combinatorica*, vol. 6, no. 1, pp. 1–13, 1986.

[7] A. H. Banihashemi and I. F. Blake, "Trellis complexity and minimal trellis diagrams of lattices," *IEEE Trans. Inform. Theory*, vol. 44, pp. 1829–1847, Sept. 1998.

[8] A. H. Banihashemi and A. K. Khandani, "On the complexity of decoding lattices using the Korkin–Zolotarev reduced basis," *IEEE Trans. Inform. Theory*, vol. 44, pp. 162–171, Jan. 1998.

[9] J. Boutros, E. Viterbo, C. Rastello, and J.-C. Belfiore, "Good lattice constellations for both Rayleigh fading and Gaussian channels," *IEEE Trans. Inform. Theory*, vol. 42, pp. 502–518, Mar. 1996.

[10] J. H. Conway and N. J. A. Sloane, "A fast encoding method for lattice codes and quantizers," *IEEE Trans. Inform. Theory*, vol. IT-29, pp. 820–824, Nov. 1983.

[11] J. H. Conway and N. J. A. Sloane, "On the Voronoi regions of certain lattices," *SIAM Journal on Algebraic and Discrete Methods*, vol. 5, pp. 294–305, Sept. 1984.

[12] J. H. Conway and N. J. A. Sloane, *Sphere Packings, Lattices and Groups.* New York, NY: Springer-Verlag, 3rd ed., 1999.

[13] R. R. Coveyou and R. D. MacPherson, "Fourier analysis of uniform random number generators," *Journal of the Association for Computing Machinery*, vol. 14, pp. 100–119, Jan. 1967.

[14] U. Dieter, "How to calculate shortest vectors in a lattice," *Mathematics of Computation*, vol. 29, pp. 827–833, Jul. 1975.

[15] T. Eriksson and E. Agrell, "On separation of linear phase from arbitrary phase vectors," unpublished work, 1999.

[16] U. Fincke and M. Pohst, "Improved methods for calculating vectors of short length in a lattice, including a complexity analysis," *Mathematics of Computation*, vol. 44, pp. 463–471, Apr. 1985.

[17] G. D. Forney, Jr., "Coset codes—part II: Binary lattices and related codes," *IEEE Trans. Inform. Theory*, vol. 34, pp. 1152–1187, Sept. 1988.

[18] O. Goldreich, D. Micciancio, S. Safra, and J. P. Seifert, "Approximating shortest lattice vectors is not harder that approximating closest lattice vectors," *Information Processing Letters*, vol. 71, pp. 55–61, July 1999.

[19] W. W. Hager, *Applied Numerical Linear Algebra*. Englewood Cliffs, NJ: Prentice Hall, 1988.

[20] B. Helfrich, "Algorithms to construct Minkowski reduced and Hermite reduced lattice bases," *Theoretical Computer Science*, vol. 41, nos. 2–3, pp. 125–139, 1985.

[21] M. Henk, "Note on shortest and nearest lattice vectors," *Information Processing Letters*, vol. 61, pp. 183–188, 1997.

[22] R. Kannan, "Improved algorithms for integer programming and related lattice problems," in *Proc. of the ACM Symposium on Theory of Computing*, (Boston, MA), pp. 193–206, Apr. 1983.

[23] R. Kannan, "Minkowski's convex body theorem and integer programming," *Mathematics of Operations Research*, vol. 12, pp. 415–440, Aug. 1987.

[24] A. K. Khandani and M. Esmaeili, "Successive minimization of the state complexity of the self-dual lattices using Korkin-Zolotarev reduced basis," Tech. Rep. UW-E&CE#97-01, Dept. of Electrical and Computer Engineering, Univ. of Waterloo, Waterloo, Ontario, Canada, Jan. 1997.

[25] D. E. Knuth, *The Art of Computer Programming*, vol. 2, *Seminumerical Algorithms*. Reading, MA: Addison-Wesley, 2nd ed., 1981.

[26] A. Korkine and G. Zolotareff, "Sur les formes quadratiques," *Mathematische Annalen*, vol. 6, pp. 366–389, 1873 (in French).

[27] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász, "Factoring polynomials with rational coefficients," *Mathematische Annalen*, vol. 261, pp. 515–534, 1982.

[28] D. Micciancio, "The shortest vector in a lattice is hard to approximate to within some constant," in *Proc. 39th Annual Symposium on Foundations of Computer Science*, (Palo Alto, CA, USA), pp. 92–98, Nov. 1998.

[29] H. Minkowski, *Geometrie der Zahlen*. Leipzig, Germany, 1896 (in German).

[30] H. Minkowski, *Gesammelte Abhandlungen*, vol. 2. Leipzig, Germany: Teubner, 1911 (in German).

[31] W. H. Mow, "Maximum likelihood sequence estimation from the lattice viewpoint," *IEEE Trans. Inform. Theory*, vol. 40, pp. 1591–1600, Sept. 1994.

[32] M. Pohst, "On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications," *ACM SIGSAM Bulletin*, vol. 15, pp. 37–44, Feb. 1981.

[33] S. S. Ryshkov and E. P. Baranovskii, "Classical methods in the theory of lattice packings," *Uspekhi Matematicheskikh Nauk*, vol. 34, pp. 3–64, July–Aug. 1979 (in Russian). Translated in *Russian Mathematical Surveys*, vol. 34, no. 4, pp. 1–68, 1979.

[34] C. P. Schnorr, "A hierarchy of polynomial time lattice basis reduction algorithms," *Theoretical Computer Science*, vol. 53, nos. 2–3, pp. 201–224, 1987.

[35] C. P. Schnorr and M. Euchner, "Lattice basis reduction: improved practical algorithms and solving subset sum problems," *Mathematical Programming*, vol. 66, pp. 181–191, 1994.

[36] J. Stern, "Lattices and cryptography: An overview," in *Public Key Cryptography. Proc. Int. Workshop on Practice and Theory in Public Key Cryptography* (H. Imai and Y. Zheng, eds.), (Pacifico Yokohama, Japan), pp. 50–54, Feb. 1998.

[37] G. Strang, *Linear Algebra and Its Applications*. San Diego, CA: Harcourt Brace Jovanovich, 3rd ed., 1988.

[38] V. Tarokh and A. Vardy, "Upper bounds on trellis complexity of lattices," *IEEE Trans. Inform. Theory*, vol. 43, pp. 1294–1300, July 1997.

[39] P. van Emde Boas, "Another NP-complete partition problem and the complexity of computing short vectors in a lattice," Rep. 81-04, Mathematisch Instituut, Amsterdam, The Netherlands, Apr. 1981.

[40] E. Viterbo and E. Biglieri, "A universal decoding algorithm for lattice codes," in *Proc. GRETSI*, (Juan-les-Pins, France), pp. 611–614, Sept. 1993.

[41] E. Viterbo and E. Biglieri, "Computing the Voronoi cell of a lattice: The diamond-cutting algorithm," *IEEE Trans. Inform. Theory*, vol. 42, pp. 161–171, Jan. 1996.

[42] E. Viterbo and J. Boutros, "A universal lattice code decoder for fading channels," *IEEE Trans. Inform. Theory*, vol. 45, pp. 1639–1642, July 1997.

[43] G. Voronoï, "Nouvelles applications des paramètres continus à la théorie des formes quadratiques," *Journal für die Reine und Angewandte Mathematik*, vol. 133, pp. 97–178, 1908; vol. 134, pp. 198–287, 1908; and vol. 136, pp. 67–181, 1909 (in French).