Faster Recursions in Sphere Decoding

Arash Ghasemmehdi and Erik Agrell

***Abstract*—Most of the calculations in standard sphere decoders are redundant, in the sense that they either calculate quantities that are never used or calculate some quantities more than once. A new method, which is applicable to lattices as well as finite constellations, is proposed to avoid these redundant calculations while still returning the same result. Pseudocode is given to facilitate immediate implementation. Simulations show that the speed gain with the proposed method increases linearly with the lattice dimension. At dimension 60, the new algorithms avoid about 75 % of all floating-point operations.**

***Index Terms*—Closest point search, Fincke–Pohst, lattice, Lenstra–Lenstra–Lovász (LLL) reduction, maximum likelihood (ML) detection, multiple-input multiple-output (MIMO), Schnorr–Euchner, sphere decoder.**

I. INTRODUCTION

EVERY lattice is represented with its *generator matrix* $\boldsymbol{G}$, whose entries are real numbers. Let n and m denote the number of rows and columns of $\boldsymbol{G}$ respectively with $n \leq m$. The rows of $\boldsymbol{G}$, which are $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$, are called *basis vectors* and are assumed to be linearly independent vectors in $\mathbb{R}^m$. The lattice of dimension n is defined as the set of points

$$\Lambda(\boldsymbol{G}, \mathbb{Z}) = \{u_1\boldsymbol{b}_1 + \ldots + u_n\boldsymbol{b}_n \mid u_i \in \mathbb{Z}\}. \tag{1}$$

This paper is about methods to find the *closest point* in a lattice to a given vector $\boldsymbol{r} \in \mathbb{R}^m$, hereafter called *received vector,* which requires minimization of the metric $\|\boldsymbol{r} - \boldsymbol{u}\boldsymbol{G}\|$ over all lattice points $\boldsymbol{u}\boldsymbol{G}$ with $\boldsymbol{u} \in \mathbb{Z}^n$.

In 1981, Pohst proposed the first closest point algorithm [1], in the equivalent context of finding the shortest vector in a translated lattice. It was improved by Fincke and Pohst in 1985 [2]. The general method has later become known as *sphere decoding,* because it relies on enumerating all lattice points inside a sphere. Mow [3], [4] and Viterbo and Biglieri [5] were the first to apply the Fincke–Pohst algorithm to maximum likelihood (ML) detection in communications. In 1999, Viterbo and Boutros extended the algorithm to finite constellations [6]. Agrell *et al.* in 2002 showed that the Schnorr–Euchner (SE) enumeration strategy [7] reduces the complexity of sphere decoding compared with the Pohst enumeration [8].

During the last decade, a lot of work has been done to improve the efficiency of sphere decoder algorithms [9]–[14], due to the significant usage they have found in numerous types of applications. In communication theory, the closest point problem arises in ML detection for multiple-input multiple-output (MIMO) channels [9], [15]–[17], ML sequence estimation [4], quantization [18], vector perturbation in multiuser communications [19], and joint detection in direct-sequence multiple access system [20].

The closest point search algorithms can be modified to find the ML point in finite constellations [6], [9], which has an important application in MIMO channels. Assuming a system with n transmit and m receive antennas, the new set of points $\Lambda(\boldsymbol{G}, \mathcal{U})$ is defined by replacing $\mathbb{Z}$ in (1) with the finite range of integers

$$\mathcal{U} = \{U_{\min}, U_{\min} + 1, \ldots, U_{\max}\}. \tag{2}$$

The transmit set can be mapped to an L-PAM constellation with $L = U_{\max} - U_{\min} + 1$. The received vector after an additive white Gaussian noise (AWGN) channel with double-sided noise power spectral density $N_0/2$ is

$$\boldsymbol{r} = \boldsymbol{u}\boldsymbol{G} + \boldsymbol{n}, \tag{3}$$

where $\boldsymbol{u} \in \mathcal{U}^n$, $\boldsymbol{r} \in \mathbb{R}^m$, $\boldsymbol{G} \in \mathbb{R}^{n \times m}$, and $\boldsymbol{n} \in \mathbb{R}^m$ is a vector of independent and identically distributed (i.i.d.) Gaussian noise with variance $N_0/2$. In this case, ML detection is equivalent to minimization of the metric $\|\boldsymbol{r} - \boldsymbol{u}\boldsymbol{G}\|$ over all possible points $\boldsymbol{u}\boldsymbol{G}$ with $\boldsymbol{u} \in \mathcal{U}^n$. In MIMO systems where usually quadrature amplitude modulation (QAM) is used, the L^2-QAM signal constellation can be viewed as two real-valued L-PAM constellations with $\boldsymbol{u} \in \mathcal{U}^{2n}$, $\boldsymbol{r} \in \mathbb{R}^{2m}$, $\boldsymbol{G} \in \mathbb{R}^{2n \times 2m}$, and $\boldsymbol{n} \in \mathbb{R}^{2m}$.

For both types of applications, lattices or finite constellations, the calculations can be implemented based on $\boldsymbol{G}$, as in the original Pohst algorithm and its numerous refinements, notably [2], [4], [9], or based on $\boldsymbol{H} = \boldsymbol{G}^{-1}$ [8], [21]. Its transpose $\boldsymbol{H}^T$ is a generator matrix for the dual lattice.

In this paper, we draw attention to a hitherto unnoticed problem with the standard algorithms. It is illustrated that the standard sphere decoder algorithms based on Pohst [2], [9] and SE [7], [9] enumeration strategies perform many excessive numerical operations. A method is proposed to avoid these unnecessary computations. However, the revision proposed is not related to choosing a more accurate upper bound on $\|\boldsymbol{r} - \boldsymbol{u}\boldsymbol{G}\|$ or scanning set of feasible point $\boldsymbol{u}\boldsymbol{G}$ in a different order. We believe that the SE strategy is the best way in this regard. Our modifications instead change how lattice vectors are recursively constructed from lower-dimensional lattices (for $\boldsymbol{G}$-based implementations) or how the received vector $\boldsymbol{r}$ is recursively projected onto the basis vectors (for $\boldsymbol{H}$-based implementations), which accounts for most of the floating point calculations in sphere decoding. With the proposed methods, not a single value would be calculated twice or remain without any use. Standalone implementations of the new (and old) algorithms are given in Fig. 2.

Manuscript submitted May 2009; revised May 2010 and Dec. 2010.

A. Ghasemmehdi was with the Department of Signals and Systems, Chalmers University of Technology, SE-41296 Göteborg, Sweden. He is now with the Department of Electrical and Computer Engineering, University of Toronto, 10 Kings College Road, Toronto, Ontario M5S 3G4, Canada (e-mail: arash.ghasemmehdi@utoronto.ca).

E. Agrell is with the Dept. of Signals and Systems, Chalmers University of Technology, SE-41296 Göteborg, Sweden (e-mail: agrell@chalmers.se).

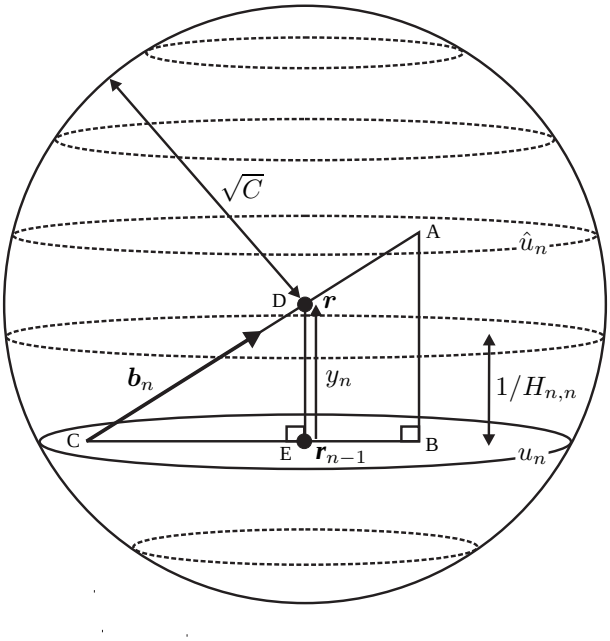

Fig. 1. Snapshot of an n-dimensional hypersphere, divided into a stack of $(n-1)$-dimensional layers.

II. CLOSEST POINT SEARCH ALGORITHMS

Without loss of generality, we assume that $\boldsymbol{G}$ is a square lower-triangular matrix with positive diagonal elements [8]. Consequently, $\boldsymbol{H} = \boldsymbol{G}^{-1}$ is also square and lower triangular with positive diagonal elements. We denote with G_{ij} and H_{ij} the element of $\boldsymbol{G}$ and $\boldsymbol{H}$, resp., at row i and column $j \leq i$. The description of the sphere decoding principle in this section takes the $\boldsymbol{H}$-based approach.

Every lattice can be divided into layers of lower-dimensional lattices. The diagonal elements $G_{ii} = 1/H_{ii}$ equal the distances between the $(i-1)$-dimensional layers in an i-dimensional layer.

Fig. 1 illustrates an n-dimensional hypersphere with radius $\sqrt{C}$ centered on a vector $\boldsymbol{r}$. All lattice points inside this hypersphere lie on $(n-1)$-dimensional layers, enumerated by the integer u_n. The basis vector $\boldsymbol{b}_n$ is in the same direction as the hypotenuse of the right triangles $\triangle$ABC and $\triangle$DEC, while all the other basis vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-1}$ lie in the subspace spanned by one of these $(n-1)$-dimensional layers.

Starting from dimension n, the received vector $\boldsymbol{r} = (r_1, r_2, \ldots, r_n) \in \mathbb{R}^n$ is projected onto the lattice basis vectors $\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_n$. This is done by a simple matrix multiplication $\boldsymbol{e}_n\boldsymbol{G} = \boldsymbol{r} \Rightarrow \boldsymbol{e}_n = \boldsymbol{r}\boldsymbol{H}$, where $\boldsymbol{e}_n = (E_{n,1}, E_{n,2}, \ldots, E_{n,n}) \in \mathbb{R}^n$. For each $(n-1)$-dimensional layer u_n that is to be examined, the orthogonal displacement y_n from the received vector $\boldsymbol{r}$ to this layer is calculated, which is shown with the line $\overline{\mathrm{DE}}$ in Fig. 1. This displacement follows from the congruence of $\triangle$ABC and $\triangle$DEC:

$$\frac{(\hat{u}_n - u_n)\frac{1}{H_{n,n}}}{y_n} = \frac{(\hat{u}_n - u_n)\|\boldsymbol{b}_n\|}{(E_{n,n} - u_n)\|\boldsymbol{b}_n\|} \Rightarrow$$
$$y_n = \frac{E_{n,n} - u_n}{H_{n,n}}. \tag{4}$$

The possible range of the layer index u_n follows from $|y_n| \leq \sqrt{C}$, which yields

$$\lceil -H_{n,n}\sqrt{C} + E_{n,n} \rceil \leq u_n \leq \lfloor H_{n,n}\sqrt{C} + E_{n,n} \rfloor, \tag{5}$$

where $\lceil\ \rceil$ and $\lfloor\ \rfloor$ denote the round up and round down operations respectively, which is also seen in Fig. 1.

In order to calculate $E_{n-1,n-1}$, which will be used later on to calculate the range of u_{n-1} (9) and the displacement y_{n-1} (14), the received vector $\boldsymbol{r}$ is first projected onto the examined $(n-1)$-dimensional layer (6) and then to the lattice basis vectors (7). We use the notation $\boldsymbol{r}_{n-1}$ for the projected received vector $\boldsymbol{r}$, where $n-1$ denotes the dimension of the layer that the received vector is projected on.

Thanks to the lower-triangular representation, the orthogonal projection of $\boldsymbol{r}$ onto the $(n-1)$-dimensional layer currently being investigated affects only the last component of $\boldsymbol{r}$. Thus, it is sufficient to subtract y_n from the nth element of $\boldsymbol{r}$ to obtain

$$\boldsymbol{r}_{n-1} = (r_1, r_2, \ldots, r_n - y_n). \tag{6}$$

This positions $\boldsymbol{r}_{n-1}$ exactly on E. In analogy with $\boldsymbol{e}_n = \boldsymbol{r}\boldsymbol{H}$, the projection of $\boldsymbol{r}_{n-1}$ onto the basis vectors is

$$\begin{aligned} \boldsymbol{e}_{n-1} &= \boldsymbol{r}_{n-1}\boldsymbol{H} && (7)\\ &= \boldsymbol{r}\boldsymbol{H} - (0, \ldots, 0, y_n)\boldsymbol{H} \\ &= \boldsymbol{e}_n - y_n(H_{n,1}, \ldots, H_{n,n}), && (8) \end{aligned}$$

where $\boldsymbol{e}_{n-1} = (E_{n-1,1}, \ldots, E_{n-1,n-1}, u_n)$. The important element here is $E_{n-1,n-1}$, because it determines the corresponding range for u_{n-1}

$$\begin{aligned} \lceil -H_{n-1,n-1}\sqrt{C - y_n^2} + E_{n-1,n-1} \rceil &\leq u_{n-1} \\ &\leq \lfloor H_{n-1,n-1}\sqrt{C - y_n^2} + E_{n-1,n-1} \rfloor, \end{aligned} \tag{9}$$

which follows from $y_{n-1}^2 + y_n^2 \leq C$ where $y_{n-1} = (E_{n-1,n-1} - u_{n-1})/H_{n-1,n-1}$.

The sphere decoder is applied recursively to search this $(n-1)$-dimensional layer. Thereafter the next u_n value in (5) is generated and a new $(n-1)$-dimensional layer is searched. Generalizing, the closest point in an i-dimensional layer is found by dividing the layer into $(i-1)$-dimensional layers, searching each of these separately, and then proceeding to the next i-dimensional layer. We will refer to this process of decreasing and increasing i as *moving down and up the layers*, resp. The projection of $\boldsymbol{r}$ onto an i-dimensional layer, where $0 \leq i \leq n-1$, is

$$\boldsymbol{r}_i = (r_1, \ldots, r_i, r_{i+1} - y_{i+1}, \ldots, r_n - y_n) \tag{10}$$

which differs from $\boldsymbol{r}_{i+1}$ in one coordinate only. The coefficients of $\boldsymbol{r}_i$ expressed as a linear combination of the basis vectors are

$$\begin{aligned} \boldsymbol{e}_i &= \boldsymbol{r}_i\boldsymbol{H} && (11)\\ &= \boldsymbol{e}_n - \sum_{j=i+1}^{n} y_j(H_{j,1}, \ldots, H_{j,n}) \\ &= \boldsymbol{e}_{i+1} - y_{i+1}(H_{i+1,1}, \ldots, H_{i+1,n}) && (12)\\ &= (E_{i,1}, \ldots, E_{i,i}, u_{i+1}, \ldots, u_n). && (13) \end{aligned}$$

(In a zero-dimensional layer, which is a lattice point, $\boldsymbol{r}_0 \in \Lambda(\boldsymbol{G}, \mathbb{Z})$ and $\boldsymbol{e}_0 = \boldsymbol{r}_0 \boldsymbol{H} \in \mathbb{Z}^n$.)

Assuming an i-dimensional sphere similar to Fig. 1, the orthogonal displacement between the projected vector $\boldsymbol{r}_i$ and the examined $(i-1)$-dimensional layer is, for any $1 \leq i \leq n$,

$$y_i = \frac{E_{i,i} - u_i}{H_{i,i}}. \tag{14}$$

Finally, the range of u_i for $i = 1, \ldots, n-1$ is determined by the *projection value* $E_{i,i}$ as

$$\lceil -H_{i,i}\sqrt{C - \lambda_{i+1}} + E_{i,i} \rceil \leq u_i \leq \lfloor H_{i,i}\sqrt{C - \lambda_{i+1}} + E_{i,i} \rfloor, \tag{15}$$

where

$$\lambda_i = y_i^2 + y_{i+1}^2 + \ldots + y_n^2 \tag{16}$$

is the squared distance from the received vector $\boldsymbol{r}$ to the projected vector $\boldsymbol{r}_{i-1}$ and $C - \lambda_{i+1}$ is the squared radius of the examined i-dimensional layer. Hence, λ_1 denotes the Euclidean distance between the received vector $\boldsymbol{r}$ and a potential closest point $\boldsymbol{r}_0$.

III. AVOIDING REDUNDANT CALCULATIONS

In this section, we claim that most of the arithmetic operations in standard sphere decoders are redundant and we propose methods to avoid them, thus increasing the decoding speed. The redundant operations are of two types: for $\boldsymbol{H}$-based implementations, numerous quantities are calculated which are never used, and for $\boldsymbol{G}$-based implementations, some quantities are calculated more than once. In both cases, the source of the problem is the way the projection values are calculated.

In Sec. IV, we demonstrate by simulations how the computational complexity of sphere decoder algorithms, for both lattices and finite constellations, is reduced due to the proposed methods. A small penalty is paid in terms of memory usage and memory write operations.

A. $\boldsymbol{H}$-Based Decoding: Projection of The Received Vector

Most of the numerical operations carried out in standard sphere decoders based on $\boldsymbol{H}$ are related to the projection of the received vector $\boldsymbol{r}$, or its lower-dimensional counterpart, onto the lattice basis vectors as in (11)–(13). Defining a matrix $\boldsymbol{E}'$ whose rows are $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n$, it follows from (12) that all elements of this matrix are updated from the elements immediately below. However, the only elements that are required in (14) and (15) are the diagonal elements $E_{i,i}$. The elements located above the diagonal of $\boldsymbol{E}'$ are equal to values of u_j that have already been calculated in previous stages of the algorithm, see (13). We therefore define a matrix $\boldsymbol{E}$ as the lower-triangular part (including the diagonal) of $\boldsymbol{E}'$.

The sphere decoder proposed in [8] always updates the first i elements of $\boldsymbol{e}_i$ simultaneously, which we call a *row-wise* updating strategy. For instance, if move down to an i-dimensional layer, we update $E_{i,j}$ for all $j = 1, \ldots, i$. These values may be used later to update $E_{j,j}$ for some $j < i$ after moving further down the layers. But why should one project the entire vector $\boldsymbol{r}_i$ onto the lattice basis vectors and calculate $E_{i,j}$ for all $j = 1, \ldots, i-1$, when they are not supposed to be used at that stage of the algorithm, and possibly not at all? The answer to this question inspired an intelligent algorithm to manage the projection of $\boldsymbol{r}$ and updating the $E_{j,i}$ values, based on following criteria:

- The last row of $\boldsymbol{E}$, $\boldsymbol{e}_n$, is calculated just once since there exists only a single n-dimensional layer.
- When moving *down* to an i-dimensional layer, $E_{i,i}$ needs to be calculated. This is done by first calculating those values of $E_{j,i}$ for $j > i$ that are not already known, which we call a *column-wise* updating strategy.
- When moving *up* from an i-dimensional layer, the first $i+1$ elements of $\boldsymbol{e}_i$ and of the other $\boldsymbol{e}$ vectors above that row become obsolete and should be considered as unknown in the future, since the next time the algorithm will move down to dimension i, the received vector $\boldsymbol{r}$ will be projected onto *another* i-dimensional layer. However, the elements below $\boldsymbol{e}_i$ remain unaffected.
- In the progression down and up the layers, the algorithm needs to keep track of which elements of $\boldsymbol{E}$ are known, to avoid recalculating them. We introduce the integers $d_1, \ldots, d_n$ for this purpose, defined such that at any instant, $E_{j,i}$ is known for $i = 1, \ldots, n$ and $j = d_i, d_i + 1, \ldots, n$. Hence, when moving down to an i-dimensional layer, the row-wise updating strategy (12) can be replaced by the more efficient column-wise updating strategy

$$E_{j-1,i} = E_{j,i} - y_j H_{j,i} \tag{17}$$

for $j = d_i, d_i - 1, \ldots, i+1$.

B. $\boldsymbol{G}$-Based Decoding: Updating The Projection Values

Also in the $\boldsymbol{G}$-based implementations, the time-consuming step is to calculate the projection values, which we denote with $E_{i,i}$ in $\boldsymbol{H}$-based implementations, as discussed in Sec. III-A, and p_i in $\boldsymbol{G}$-based implementations.

According to [9], which uses the same recursions as [6], the projection value when moving down to an i-dimensional layer is calculated as $p_i = f_i / G_{i,i}$, where $f_n = r_n$ and

$$f_i = r_i - \sum_{k=i+1}^{n} u_k G_{k,i}. \tag{18}$$

In contrast to (12), this is a column-wise updating strategy, but not the most efficient one. Moving further down the layers in order to calculate p_j for $j < i$, one can notice that part of the sum in (18) is already calculated and does not need to be recalculated if stored in memory. Hence, we define $F_{j,i} = r_i - \sum_{k=j+1}^{n} u_k G_{k,i}$ for $1 \leq i \leq j < n$ and $F_{n,i} = r_i$ for $1 \leq i \leq n$. As a result, (18) is equivalent to calculating

$$F_{j-1,i} = F_{j,i} - u_j G_{j,i} \tag{19}$$

for $j = n, n-1, \ldots, i+1$ and then $p_i = F_{i,i} / G_{i,i}$. If some values of $F_{j,i}$ are already known, however, the recursion (19) can begin at some $j < n$, which saves operations compared with (18).

We collect the elements $F_{j,i}$ in a lower-triangular matrix $\boldsymbol{F}$, which is related to the lower-triangular matrix $\boldsymbol{E}$ by $\boldsymbol{E} = \boldsymbol{F}\boldsymbol{H}$ or, equivalently, $\boldsymbol{F} = \boldsymbol{E}\boldsymbol{G}$. The optimized projection method

proposed in Sec. III-A to update the $E_{j,i}$ values, with some minor modifications, can be similarly applied to update $F_{j,i}$. The changes are as follows:

- The last row of $\boldsymbol{F}$ is equal to the received vector $\boldsymbol{r}$.
- When moving *down* to an i-dimensional layer, $F_{i,i}$ needs to be calculated. This is done by computing (19) for $j = d_i, d_i - 1, ..., i + 1$, where d_i, in analogy with the definition above for $\boldsymbol{H}$-based decoding, is defined as the minimum j for which $F_{j,i}$ is known.
- When moving *up* from an i-dimensional layer, all elements on the ith row of $\boldsymbol{F}$ and above that row become obsolete and should be considered as unknown in the future, since the next time the algorithm will move down to dimension i, another i-dimensional layer will be investigated. However, the elements below that row remain unaffected.

C. The Proposed Algorithm

Standalone representations of the old and new algorithms, $\boldsymbol{G}$-based and $\boldsymbol{H}$-based versions, for lattices and finite constellations, are given in Fig. 2, all based on the SE enumeration strategy. The specifications are intended to be sufficiently detailed to allow a straightforward implementation, even without knowledge of the underlying theory.

As starting points, we use the $\boldsymbol{G}$-based algorithm called "Algorithm II" in [9], labeled with 2 in Fig. 2, and the $\boldsymbol{H}$-based algorithm "Decode" in [8], here labeled with 3. The loops have been restructured for consistency between the algorithms, but the calculations in Fig. 2 are exactly the same as in [9] and [8]. Indeed, all algorithms for lattice decoding (algorithms 1, 3, 5, and 7) visit the same layers u_i, in the same order, and return the same result $\hat{\boldsymbol{u}}$, although they calculate different intermediate quantities. A similar note holds for decoding finite constellations (algorithms 2, 4, 6, and 8).

After the initialization, the algorithms are divided into three parts. In the first part, which is the first ***do–while*** loop, we move down the layers (decrease i) as long as the squared Euclidean distance λ_i (16) between the received vector $\boldsymbol{r}$ and the projected vector $\boldsymbol{r}_{i-1}$ (10) is less than the squared Euclidean distance C between the received vector $\boldsymbol{r}$ and the closest lattice point detected so far. In the second part, which is the second ***do–while*** loop, we move up in the hierarchy of layers (increase i) as long as $\lambda_i \geq C$. Moreover, before leaving each of these parts, we store the minimum and maximum dimension i that has been visited (in the variables m and i, respectively). These values are used in the last part of the algorithm, which only belongs to the new algorithms.

The method to manage the recursive projection of $\boldsymbol{e}_i$ or the calculation of $F_{j,i}$ is proposed in the third and last part, after the second ***while***. This part sets the value of d_j to i for $j = m, m+1, \ldots, i-1$ and $\max\{d_j, i\}$ for $j = 1, \ldots, m-1$, but in a way that generally requires fewer than $m - 1$ integer comparisons. The values of d_j for $j \geq i$ are not updated, since they will not be needed in the next iteration. The values d_j keep track of which elements $E_{j,i}$ or $F_{j,i}$ are known at each instant, which in turn determines the range of j values for which (17) and (19) should be computed, as detailed in Sec. III-A

```
1 5      input: n, G, r; output: û ∈ Z^n
 2 6     input: n, G, r, U_min, U_max; output: û ∈ U^n
  3 7    input: n, H, r; output: û ∈ Z^n
   4 8   input: n, H, r, U_min, U_max; output: û ∈ U^n
12345678 C = ∞
12 5678  i = n + 1
  34     i = n
    5678 d_j = n,  j = 1, ..., n
12345678 λ_{n+1} = 0
  34 78  E_{n,j} = Σ_{k=j}^{n} r_k H_{k,j},  j = 1, ..., n
    56   F_{n,j} = r_j,  j = 1, ..., n
 3       u_n = round(E_{n,n})
  4      u_n = roundc(E_{n,n})
  34     y = (E_{n,n} − u_n)/H_{n,n}
  34     Δ_n = sign(y)
  34     λ_n = y^2
12345678 LOOP
12345678 do {
12345678   if (i ≠ 1) {
12345678     i = i − 1
  34         E_{i,j} = E_{i+1,j} − y H_{i+1,j},  j = 1, ..., i
    56       F_{j−1,i} = F_{j,i} − u_j G_{j,i},  j = d_i, d_i − 1, ..., i + 1
      78     E_{j−1,i} = E_{j,i} − y_j H_{j,i},  j = d_i, d_i − 1, ..., i + 1
12           p_i = (r_i − Σ_{j=i+1}^{n} u_j G_{j,i})/G_{i,i}
    56       p_i = F_{i,i}/G_{i,i}
1 5          u_i = round(p_i)
 2 6         u_i = roundc(p_i)
  3 7        u_i = round(E_{i,i})
   4 8       u_i = roundc(E_{i,i})
12 56        y = (p_i − u_i)G_{i,i}
  34         y = (E_{i,i} − u_i)/H_{i,i}
      78     y_i = (E_{i,i} − u_i)/H_{i,i}
123456       Δ_i = sign(y)
      78     Δ_i = sign(y_i)
123456       λ_i = λ_{i+1} + y^2
      78     λ_i = λ_{i+1} + y_i^2
12345678   } else {
12345678     û = u
12345678     C = λ_1
12345678   }
12345678 } while (λ_i < C)
    5678 m = i
12345678 do {
12345678   if (i = n)
12345678     return û and exit
12345678   else {
12345678     i = i + 1
 2 4 6       y = ∞
        8    y_i = ∞
12345678     u_i = u_i + Δ_i
12345678     Δ_i = −Δ_i − sign(Δ_i)
 2 4 6 8     if (U_min ≤ u_i ≤ U_max)
12  56         y = (p_i − u_i)G_{i,i}
  34           y = (E_{i,i} − u_i)/H_{i,i}
      78       y_i = (E_{i,i} − u_i)/H_{i,i}
 2 4 6 8     else {
 2 4 6 8       u_i = u_i + Δ_i
 2 4 6 8       Δ_i = −Δ_i − sign(Δ_i)
 2 4 6 8       if (U_min ≤ u_i ≤ U_max)
 2   6           y = (p_i − u_i)G_{i,i}
   4             y = (E_{i,i} − u_i)/H_{i,i}
       8         y_i = (E_{i,i} − u_i)/H_{i,i}
 2 4 6 8       }
123456         λ_i = λ_{i+1} + y^2
      78       λ_i = λ_{i+1} + y_i^2
12345678     }
12345678   } while (λ_i ≥ C)
    5678   d_j = i,  j = m, m + 1, ..., i − 1
    5678   for (j = m − 1, m − 2, ..., 1) {
    5678     if (d_j < i)
    5678       d_j = i
    5678     else
    5678       goto LOOP
    5678   }
12345678   goto LOOP
```

$$\operatorname{sign}(x) = \begin{cases} -1, & x \leq 0 \\ 1, & x > 0 \end{cases}$$

$$\operatorname{round}(x) = \arg\min_{u \in \mathbb{Z}} |u - x|$$

$$\operatorname{roundc}(x) = \arg\min_{u \in \mathcal{U}} |u - x|$$

n: dimension

$\boldsymbol{G}$: a lower-triangular $n \times n$ generator matrix with positive diagonal elements

$\boldsymbol{H} = \boldsymbol{G}^{-1}$

$\boldsymbol{r}$: received vector

$U_{\min}, U_{\max}$: constellation endpoints (2)

$$\hat{\boldsymbol{u}} = \arg\min_{\boldsymbol{u}} \|\boldsymbol{r} - \boldsymbol{u}\boldsymbol{G}\|$$

1 old $\boldsymbol{G}$-based, lattices
2 old $\boldsymbol{G}$-based, finite const. [9]
3 old $\boldsymbol{H}$-based, lattices [8]
4 old $\boldsymbol{H}$-based, finite const.
5 new $\boldsymbol{G}$-based, lattices
6 new $\boldsymbol{G}$-based, finite const.
7 new $\boldsymbol{H}$-based, lattices
8 new $\boldsymbol{H}$-based, finite const.

Fig. 2. Eight algorithms in one figure. To implement a certain algorithm, use only the lines labeled with the algorithm's digit 1,. . . ,8.

and III-B. It can be shown that d_j is a decreasing sequence for $j = 1, 2, \ldots, m$, constant for $j = m, m+1, \ldots, i-1$, and finally increasing for $j = i-1, i, \ldots, n$.

Due to the well-documented performance gain that the SE enumeration strategy brings to sphere decoders, we apply herein the proposed refinement only to the SE strategy. However, the same refinement can be applied to the original Pohst enumeration strategy. It is also applicable to most, or all, of the numerous sphere decoder variants, optimal as well as suboptimal, that have been developed in the last decade.

IV. SIMULATION RESULTS

Herein, we evaluate the effectiveness of the proposed smart vector projection technique on the sphere decoder algorithms based on SE enumeration strategy, for both lattices and finite constellations. All eight algorithms are implemented according to the pseudocode presented in Fig. 2.

We base our performance comparison measure on counting the number of floating point operations (flops) and integer operations (intops) that each algorithm carries out to reach the closest lattice point. Both types of operations include addition, subtraction, multiplication, division, and comparison, but not ***for*** loop counters, whose role differs between programming languages. The ***round*** operation is counted as a single floating point operation, and ***roundc*** in Sec. IV-B is counted as one floating point operation for 2-PAM and two for 4-PAM.

To compare the complexity of two algorithms, typically an old and a new one, we generate M random generator matrices $\boldsymbol{G}_1, \ldots, \boldsymbol{G}_M$, and for each $\boldsymbol{G}_j$ we generate N random received vectors $\boldsymbol{r}_{j,1}, \ldots, \boldsymbol{r}_{j,N}$. The same vectors are decoded using both algorithms and the number of operations $ops(\boldsymbol{r}_{j,i}, \boldsymbol{G}_j)$ is counted, which could be either flops or intops. The average gain with the new algorithm is reported as

$$gain = \frac{1}{M} \sum_{j=1}^{M} \frac{\sum_{i=1}^{N} ops_{\text{old}}(\boldsymbol{r}_{j,i}, \boldsymbol{G}_j)}{\sum_{i=1}^{N} ops_{\text{new}}(\boldsymbol{r}_{j,i}, \boldsymbol{G}_j)}. \tag{20}$$

A. Lattices

We generate the lattice generator matrices with random numbers, drawn from i.i.d. zero-mean, unit-variance Gaussian distributions. The random input vectors are generated uniformly inside a Voronoi region according to [22]. Our simulation results are based on averaging over $M = 100$ different generator matrices. The number of input vectors N depends on the dimension n of the lattices. Fewer input vectors are examined in high dimensions, to the extent that we ensure that the plotted curves are reasonably smooth.

Fig. 3 compares the number of flops for the standard $\boldsymbol{G}$- and $\boldsymbol{H}$-based algorithms (algorithms 1 and 3 in Fig. 2) with the new algorithms proposed in this paper (algorithms 5 and 7). Apparently, the $\boldsymbol{G}$- and $\boldsymbol{H}$-based implementations have about the same complexity, but both can be significantly improved.

The gain (20) is shown in Fig. 4 for flops and intops. A preprocessing stage was applied to each lattice, replacing the generator matrix with another generator matrix for the same lattice via the so-called Lenstra–Lenstra–Lovász (LLL) reduction [23], [24]. The operations needed for the reduction were not counted, since the preprocessing is only done once for each lattice, regardless of the number of received vectors. The gain with new algorithms increases linearly with dimension, while the reduction does not change the ratios substantially. The drawback is a somewhat larger number of intops, but the penalty converges to a mere 15% increase at high dimensions. In simulations it was observed that most of the operations in the algorithms are flops, especially as the dimension increases. For instance, at dimension 60 with the old $\boldsymbol{H}$-based algorithm, the flops are roughly 10 times more than the intops. Hence, flops dominate the complexity of the algorithms and intops have a relatively small effect on the overall complexity.

We also measured the running time for the algorithms. As expected, the gain increases roughly linearly with the dimension, similarly to the flops curves in Fig. 4. However, the slope of the curve varies significantly between different processors and compilers, which is why we did not include running time in Fig. 4. At dimension 60, the gain ranged from 1.7 (AMD processor, Visual C++ compiler) to 2.7 (Intel processor, GCC compiler), for the $\boldsymbol{H}$-based algorithm without reduction. We can thus safely conclude that the reduced number of operations translates into a substantial speed gain, but how much depends on the computer architecture.

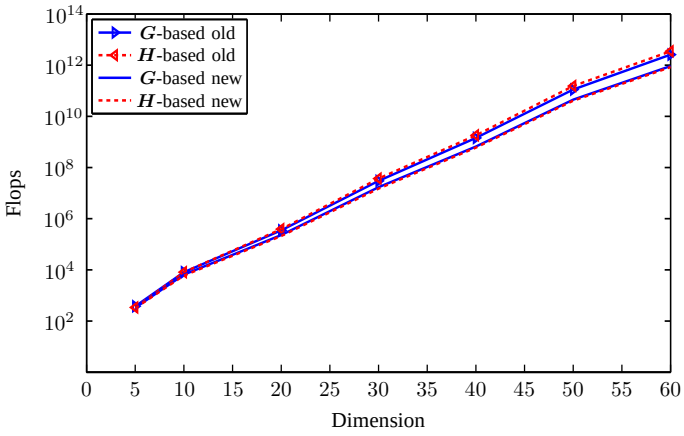

Fig. 3. The average number of flops needed to decode a vector with the old and new versions of $\boldsymbol{G}$- and $\boldsymbol{H}$-based lattice decoding algorithms, without reduction.

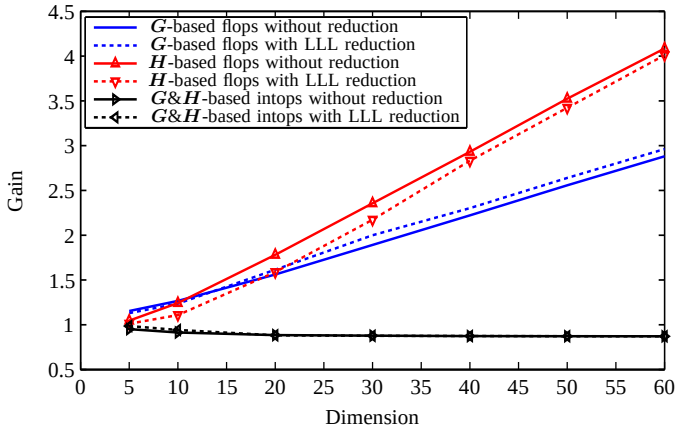

Fig. 4. Complexity gain with the new lattice decoding algorithms, with and without reduction.

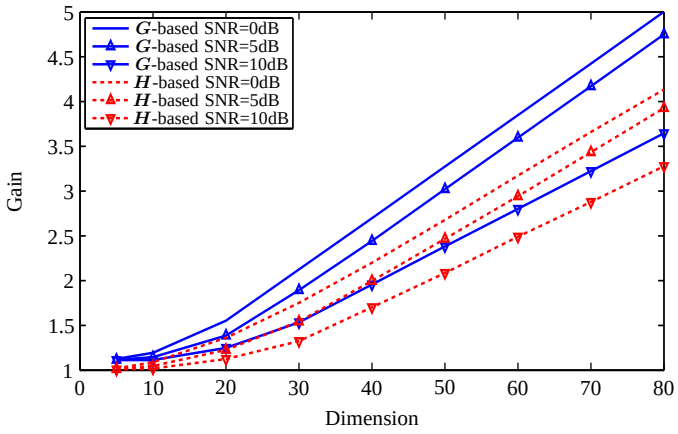

Fig. 5. Average gain in the number of flops with the new algorithms for a 2-PAM constellation and various SNRs.

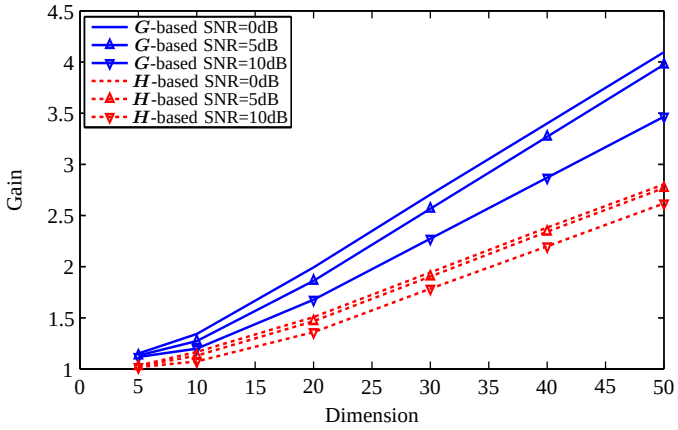

Fig. 6. Average gain in the number of flops for a 4-PAM constellation and various SNRs.

B. Finite Constellations

The channel model in (3) for an L-PAM constellation is considered, where the average symbol energy of the constellation, E_s, is calculated from the signal set $\{-\frac{L-1}{2}, -\frac{L-1}{2}+1, \ldots, \frac{L-1}{2}\}$ and the SNR is defined as E_b/N_0, where $E_b = E_s/\log_2 L$ is the average energy per bit and $N_0/2$ is the double-sided noise spectral density.

The gain in flops is presented in Figs. 5–6 for 2-PAM and 4-PAM constellations, resp., averaged over 100 random channel matrices $\boldsymbol{G}$ with i.i.d. zero-mean, unit-variance elements. The same general conclusion as for lattices holds for finite constellations too: The new algorithms provide a substantial complexity gain, and the gain increases linearly with the dimension. However, in contrast to lattice decoding, the gains are here higher for $\boldsymbol{G}$-based implementations. Furthermore, the gains increase at low SNR, and 4-PAM offers slightly higher gains than 2-PAM. Tentative investigations indicate that the gains are even higher with 8-PAM.

ACKNOWLEDGMENT

We wish to thank the two anonymous reviewers for their careful reading and insightful criticism, which helped improve the presentation substantially. Thanks to their suggestions, the paper is now a more well-rounded treatment of both $\boldsymbol{H}$- and $\boldsymbol{G}$-based algorithms, and one of them even provided a small but elegant improvement to the new $\boldsymbol{G}$-based algorithms.

REFERENCES

[1] M. Pohst, "On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications," *SIGSAM Bulletin*, vol. 15, no. 1, pp. 37–44, Feb. 1981.

[2] 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, no. 170, pp. 463–471, Apr. 1985.

[3] W. H. Mow, "Maximum likelihood sequence estimation from the lattice viewpoint," in *Proc. IEEE ICCS/ISITA*, vol. 1, Singapore, Nov. 1992, pp. 127–131.

[4] ——, "Maximum likelihood sequence estimation from the lattice viewpoint," *IEEE Trans. Inf. Theory*, vol. 40, no. 5, pp. 1591–1600, Sep. 1994.

[5] E. Viterbo and E. Biglieri, "A universal decoding algorithm for lattice codes," in *Proc. 14-ème Colloque GRETSI*, Juan-les-Pins, France, Sept. 1993, pp. 611–614.

[6] E. Viterbo and J. J. Boutros, "A universal lattice code decoder for fading channels," *IEEE Trans. Inf. Theory*, vol. 45, no. 5, pp. 1639–1642, July 1999.

[7] C. P. Schnorr and M. Euchner, "Lattice basis reduction: improved practical algorithms and solving subset sum problems," *Mathematical Programming*, vol. 66, no. 2, pp. 181–199, 1994.

[8] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger, "Closest point search in lattices," *IEEE Trans. Inf. Theory*, vol. 48, no. 8, pp. 2201–2214, Aug. 2002.

[9] M. O. Damen, H. El Gamal, and G. Caire, "On maximum-likelihood detection and the search for the closest lattice point," *IEEE Trans. Inf. Theory*, vol. 49, no. 10, pp. 2389–2402, Oct. 2003.

[10] B. Shim and I. Kang, "Sphere decoding with a probabilistic tree pruning," *IEEE Trans. Signal Process.*, vol. 56, no. 10, pp. 4867–4878, Oct. 2008.

[11] R. Gowaikar and B. Hassibi, "Statistical pruning for near-maximum likelihood decoding," *IEEE Trans. Signal Process.*, vol. 55, no. 6, pp. 2661–2675, June 2007.

[12] W. Zhao and G. B. Giannakis, "Sphere decoding algorithms with improved radius search," *IEEE Trans. Commun.*, vol. 53, no. 7, pp. 1104–1109, July 2005.

[13] ——, "Reduced complexity closest point decoding algorithms for random lattices," *IEEE Trans. Wireless Commun.*, vol. 5, no. 1, pp. 101–111, Jan. 2006.

[14] K. Su and I. J. Wassell, "A new ordering for efficient sphere decoding," in *Proc. IEEE ICC*, vol. 3, Seoul, Korea, May 2005, pp. 1906–1910.

[15] O. Damen, A. Chkeif, and J. C. Belfiore, "Lattice code decoder for space-time codes," *IEEE Commun. Lett.*, vol. 4, no. 5, pp. 161–163, May 2000.

[16] W. K. Ma, B. N. Vo, T. N. Davidson, and P. C. Ching, "Blind ML detection of orthogonal space-time block codes: efficient high-performance implementations," *IEEE Trans. Signal Process.*, vol. 54, no. 2, pp. 738–751, Feb. 2006.

[17] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner, and H. Bölcskei, "VLSI implementation of MIMO detection using the sphere decoding algorithm," *IEEE J. Solid-State Circuits*, vol. 40, no. 7, pp. 1566–1577, July 2005.

[18] E. Agrell and T. Eriksson, "Optimization of lattices for quantization," *IEEE Trans. Inf. Theory*, vol. 44, no. 5, pp. 1814–1828, Sept. 1998.

[19] B. M. Hochwald, C. B. Peel, and A. L. Swindlehurst, "A vector-perturbation technique for near-capacity multiantenna multiuser communication—Part II: perturbation," *IEEE Trans. Commun.*, vol. 53, no. 3, pp. 537–544, Mar. 2005.

[20] L. Brunel and J. J. Boutros, "Lattice decoding for joint detection in direct-sequence CDMA systems," *IEEE Trans. Inf. Theory*, vol. 49, no. 4, pp. 1030–1037, Apr. 2003.

[21] C. Windpassinger, L. Lampe, R. F. H. Fischer, and T. Hehn, "A performance study of MIMO detectors," *IEEE Trans. Wireless Commun.*, vol. 5, no. 8, pp. 2004–2008, Aug. 2006.

[22] J. H. Conway and N. J. A. Sloane, "On the Voronoi regions of certain lattices," *SIAM Journal on Algebraic and Discrete Methods*, vol. 5, no. 3, pp. 294–305, Sept. 1984.

[23] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász, "Factoring polynomials with rational coefficients," *Mathematische Annalen*, vol. 261, no. 4, pp. 515–534, 1982.

[24] W. H. Mow, "Universal lattice decoding: principle and recent advances," *Wireless Communications and Mobile Computing*, vol. 3, no. 5, pp. 553–569, 2003.

Arash Ghasemmehdi received the M.Sc degree in communication engineering in 2009 from Chalmers University of Technology, Sweden. He is now a Ph.D. candidate at the Department of Electrical and Computer Engineering, University of Toronto, Canada. His research interests are in the areas of information theory, coding theory, digital communications, and signal processing.

Erik Agrell received the M.S. degree in electrical engineering in 1989 and the Ph.D. degree in information theory in 1997, both from Chalmers University of Technology, Sweden.

From 1988 to 1990, he was with Volvo Technical Development as a Systems Analyst, and from 1990 to 1997, with the Department of Information Theory, Chalmers University of Technology, as a Research Assistant. In 1997–1999, he was a Postdoctoral Researcher with the University of Illinois at Urbana-Champaign and the University of California, San Diego. In 1999, he joined the faculty of Chalmers University of Technology, first as an Associate Professor and since 2009 as a Professor in Communication Systems. His research interests include coding, modulation, and multiplexing for fiber-optic communications, bit-interleaved coded modulation and multilevel coding, bit-to-symbol mappings in coded and uncoded systems, lattice theory and sphere decoding, and multidimensional geometry.

Prof. Agrell served as Publications Editor for IEEE Transactions on Information Theory from 1999 to 2002.