On Joint Detection and Decoding of Linear Block Codes on Gaussian Vector Channels

HARIS VIKALO   BABAK HASSIBI

Department of Electrical Engineering
California Institute of Technology, Pasadena, CA 91125
e-mail: hvikalo,hassibi@systems.caltech.edu

Abstract

Optimal receivers recovering signals transmitted across noisy communication channels employ a maximum-likelihood (ML) criterion to minimize the probability of error. The problem of finding the most likely transmitted symbol is often equivalent to finding the closest lattice point to a given point and is known to be NP-hard. In systems that employ error-correcting coding for data protection, the symbol space forms a sparse lattice, where the sparsity structure is determined by the code. In such systems, ML data recovery may be geometrically interpreted as a search for the closest point in the sparse lattice. In this paper, motivated by the idea of the "sphere decoding" algorithm of Fincke and Pohst, we propose an algorithm that finds the closest point in the sparse lattice to the given vector. This given vector is not arbitrary, but rather is an unknown sparse lattice point that has been perturbed by an additive noise vector whose statistical properties are known. The complexity of the proposed algorithm is thus a random variable. We study its expected value, averaged over the noise and over the lattice. For binary linear block codes, we find the expected complexity in closed form. Simulation results indicate significant performance gains over systems employing separate detection and decoding, yet are obtained at a complexity that is practically feasible over a wide range of system parameters.

Index Terms—Integer least-squares problem, sphere decoding, wireless communications, multiple-antenna systems, lattice problems, NP hard, expected complexity, joint detection and decoding

1 Introduction

To protect transmitted information from the adverse effects of a channel, communication systems typically employ some form of error-correcting coding. On vector channels, the resulting coded word is first modulated onto symbols and then transmitted across the channel in blocks. Optimal receivers, designed to recover transmitted information so that the probability of error is minimized, should employ a maximum-likelihood (ML) criterion. However, for block transmission over Gaussian vector channels, the computational complexity of the optimal receivers is considered to be practically infeasible. In fact, the ML criterion is often thought of as one leading to an exhaustive search over the space of information vectors, which requires testing a number of hypothesis that is exponential in the dimension of the search space. To this end, heuristic techniques which have manageable complexity but sub-optimal performance are often used in practice. Moreover, to further alleviate the computational burden, the symbol detection problem is often treated separately from the data decoding. However, the bit-error-rate (BER) performance of receivers which employ detection and decoding in separate stages is, in general, inferior to those that employ them jointly. To overcome these performance losses, soft decoding techniques use probabilistic information at the output of the first stage (i.e., use soft information about the detected symbols) as the input to the second stage – the decoder. The subsequent iterative exchange of the soft information between the receiver's stages attempts to extract all the information about the original uncoded message that is contained in the received signal.

When a symbol point belongs to a lattice, ML decoding is equivalent to the search for the closest lattice point to the given (received) vector. There exist techniques that solve the closest-point search without actually performing an exhaustive search over the entire lattice (e.g., see [1] and the references therein). These techniques have recently been proposed for ML detection on (uncoded) vector Gaussian channels. In [2], the sphere decoding algorithm [3] was proposed for the decoding of lattice codes and in [4] for detection in multiple-antenna wireless communication systems. The sphere decoding algorithm finds the closest point in a lattice to the received vector but limits the search to only those lattice points that fall within a sphere centered at the received vector. In [5, 6] it was shown that when the radius of the sphere is chosen according to the noise power, the complexity of sphere decoding is random variable with a mean that is a low-degree polynomial over a wide range of signal-to-noise rations (SNR) and system dimensions. These complexity results imply practical feasibility of sphere decoding and raise the question of whether similar ideas may extend to the receiver design in systems which employ error-correcting codes.[1]

In this paper, we consider the joint ML detection and decoding on Gaussian vector channels, where the transmitted data is encoded by a linear block error-correcting code. The coded data is first modulated onto symbols, which are in this paper assumed to be points in a rectangular lattice, and then transmitted across the channel. Thus the set of all possible symbols forms a subset of the lattice, where the structure

[1]We note that sphere decoding and its extensions have already been employed for *iterative* detection and decoding in systems employing channel codes [9, 10]; however, in this paper we are concerned with *direct*, i.e. non-iterative joint detection and decoding.

and the cardinality of the subset is determined by the error-correcting code. The joint maximum-likelihood detection and decoding may hence be geometrically interpreted as a search for the closest point in a sparse lattice. Motivated by the sphere decoding idea, we propose an algorithm that finds the closest point in the lattice to the received vector by searching for the lattice points inside the sphere centered at the received vector. However, compared to the standard sphere decoding, an important additional constraint is imposed: the admissible lattice points (i.e., possible solutions) must not only be inside the sphere but also have to be valid codewords. [Clearly, the algorithm essentially performs soft decision decoding on Gaussian vector channels.]

We note that the received vector is not an arbitrary point in space but is a sparse lattice point perturbed by the Gaussian noise. Thus it is meaningful to choose the radius of the sphere according to the statistics of the noise. In particular, we choose the radius to be a linear function of the power of the noise. The computational complexity of the algorithm is a data-dependent random variable – it depends on the transmitted symbol and the particular instantiations of the channel and noise. We quantify it by means of its first moment – the expected complexity. For binary linear block codes, we find an analytic expression for the expected complexity in a closed form.

The paper is organized as follows. In Section 2, we introduce the system model and state the problem. The algorithm for the joint ML detection and decoding (the JDD-ML algorithm) for linear block codes is presented in Section 3. In Section 4, we consider the computational complexity of the algorithm and calculate the expected complexity for binary linear block codes. In Section 5, we consider cyclic codes, and Section 6 contains a description of an extension of the algorithm to the joint maximum a posteriori (MAP) detection and decoding (the JDD-MAP algorithm). Simulation results are presented in Section 7; discussion and conclusions are in Section 8.

Some of the results discussed in this paper were preliminary reported in [7].

2 System Model and Problem Statement

We consider digital communication over the Gaussian vector channel shown in Figure 1. The channel encoder in Figure 1 encodes the $k \times 1$ information data vector $\mathbf{b}$ to obtain the $m \times 1$ codeword $\mathbf{c}$. We assume that the encoder employs the block channel code defined via its generator matrix $\mathbf{G}$, i.e., the encoding

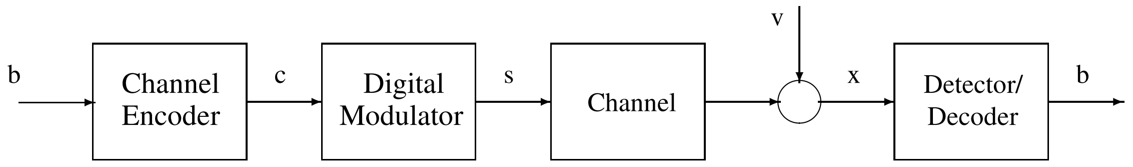

Figure 1: System Model

operation is given by[2]

$$\mathbf{c} = \mathbf{G}^T \cdot \mathbf{b}. \tag{1}$$

The size of the generator matrix $\mathbf{G}$ is $k \times m$, i.e., the rate of the code is $R_c = k/m$. The entries in $\mathbf{c}$, $\mathbf{b}$, and $\mathbf{G}$, are assumed to be elements from a Galois field $GF(L)$, where L is a power of 2. Operation "$\cdot$" in (1) denotes multiplication over the Galois field, i.e., multiplication modulo L.

As implied by Figure 1, the coded vector $\mathbf{c}$ is modulated and the resulting symbol vector $\mathbf{s}$ is then transmitted across the channel. We assume an L-PAM modulation, i.e., that each entry of the symbol vector $\mathbf{s}$ takes one of the L possible values from the set

$$\mathcal{Z}_L = \{-\frac{L-1}{2}, \dots, -\frac{1}{2}, \frac{1}{2}, \dots, \frac{L-1}{2}\}.$$

Therefore, the m-dimensional transmitted symbol vector $\mathbf{s}$ is a point in the rectangular lattice $\mathcal{Z}_L^m$. However, in the communication setup that we just described, not all points from the lattice $\mathcal{Z}_L^m$ may be transmitted. In fact, only those lattice points that may be obtained by modulating valid codewords constitute the symbol space. Therefore, the symbol space $\mathcal{D}_L^m$ is a subset of the lattice $\mathcal{Z}_L^m$, i.e., $\mathcal{D}_L^m \subset \mathcal{Z}_L^m$, and is determined by the channel code.

We choose the size of the PAM constellation $\mathcal{Z}_L$ to be as same as the size of the Galois field for simplicity (generalizations are straightforward).

For convenience, henceforth we shall denote the modulation operation by $[\cdot]$, i.e., the fact that the point $\mathbf{s}$ is obtained by modulating the code vector $\mathbf{c} = \mathbf{G}^T \cdot \mathbf{b}$ onto the L-PAM constellation will be denoted by

$$\mathbf{s} \triangleq [\mathbf{G}^T \cdot \mathbf{b}].$$

[2]In literature, the encoding operation is often defined as $\mathbf{c}^T = \mathbf{b}^T \cdot G$. We use the alternative form (1) since it proves to be more convenient for the implementation of the decoding technique that we propose later in the paper.

We assume a real-valued Gaussian vector channel model of the form

$$\mathbf{x} = H\mathbf{s} + \mathbf{v}, \tag{2}$$

where $H \in \mathcal{R}^{n \times m}$ is an equivalent channel matrix with i.i.d. Gaussian entries, and $\mathbf{v} \in \mathcal{R}^{n \times 1}$ is a noise vector with i.i.d. $\mathcal{N}(0, 1)$ entries. [Note that model (2) also describes MIMO systems which employ certain space-time codes, e.g., linear-dispersive (LD) space-time codes [8]. There, matrix H in (2) is a function of both the channel matrix and the LD code.]

The receiver that performs *joint ML detection and decoding* solves the optimization problem

$$\max_{\mathbf{b} \in GF(L)^k} p(\mathbf{x}|\mathbf{b}). \tag{3}$$

For the model (2) and the Gaussian noise,

$$\arg \max_{\mathbf{b} \in GF(L)^k} p(\mathbf{x}|\mathbf{b}) = \arg \min_{\mathbf{b} \in GF(L)^k} \|\mathbf{x} - H[\mathbf{G}^T \cdot \mathbf{b}]\|^2,$$

which transforms (3) to the equivalent problem

$$\min_{\mathbf{b} \in GF(L)^k} \|\mathbf{x} - H[\mathbf{G}^T \cdot \mathbf{b}]\|^2. \tag{4}$$

Geometrically, the integer-least squares problem (4) can be interpreted as a search for the lattice point closest to the given vector. The space over which we optimize is the information vector space $GF(L)^k$. Alternatively, we can think of it as the search over the (sparse) subset $\mathcal{D}_L^m$ of the integer lattice $\mathcal{Z}_L^m$. One way of obtaining the solution to (4) is by means of an exhaustive search over $GF(L)^k$ (or, equivalently, search over $\mathcal{D}_L^m$). However, expanding on the basic idea of the Fincke-Pohst approach [3], we propose an efficient alternative to the exhaustive search: the algorithm performs the optimization by searching only over those points in $\mathcal{D}_L^m$ that belong to a hypershere around the observed point $\mathbf{x}$.

The closest point search in the sparse lattice is illustrated in Figure 2. The blank points in Figure 2 denote the points in $H \cdot \mathcal{Z}_L^m$ (the channel-generated full lattice) that are not in $H \cdot \mathcal{D}_L^m$ (the channel-generated sparse lattice), i.e., denote the set $H \cdot \mathcal{Z}_L^m \backslash H \cdot \mathcal{D}_L^m$. Note that in Figure 2, the closest point in $H \cdot \mathcal{Z}_L^m$ to the received vector $\mathbf{x}$ is actually not in $H \cdot \mathcal{D}_L^m$. Recall that the original sphere decoding algorithm of Fincke and Pohst solves the closest-point search in the full lattice $\mathcal{Z}_L^m$. The major difference between the Fincke-Pohst

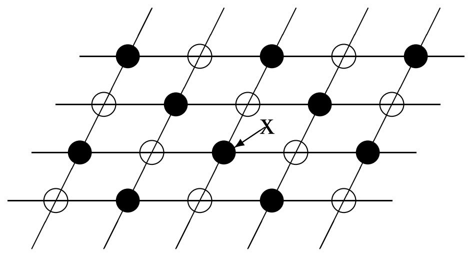

Figure 2: Closest-point search in the sparse lattice

algorithm and the algorithm that we study in this paper is the additional constraint that the possible solutions must not only be inside the hypersphere but also be valid codewords.

3 The JDD-ML algorithm

The sphere decoding (SD) algorithm solves the ML detection problem in uncoded systems by finding lattice points (in $\mathcal{Z}_L^m$ lattice) inside a sphere of a carefully chosen radius r, centered at the received vector. As discussed in [5], the algorithm achieves this by searching for lattice points inside spheres of radius r and dimensions $i = 1, 2, \ldots, m$. In this way, the algorithm essentially finds, one-by-one, all components of the lattice points inside the sphere. This is made feasible by breaking down the single condition that the lattice point be inside the sphere into a set of conditions (inequalities) that the components (i.e., coordinates) of the lattice point must satisfy in order that the point belong to the sphere. The algorithm effectively performs a tree search where the nodes in the tree correspond to the components of the unknown vector and where violating the aforementioned conditions results in tree pruning.

Motivated by the idea of SD outlined above, in this section we develop an algorithm that solves (4) by finding valid codewords inside the sphere centered at the received vector. To facilitate efficient search, we need to state the set of conditions on the coordinates of a lattice point so that, when all of such conditions are satisfied, the lattice point both belongs to the sphere and is valid codeword, i.e., can be expressed as $[\mathbf{G}^T \cdot \mathbf{b}]$ for some $\mathbf{b} \in GF(L)^k$. The search should clearly be performed over the space of information vectors, $GF(L)^k$. Note that the algorithm may return more than one solution. In fact, the algorithm generally returns a set of vectors $\mathbf{b}$ such that $[\mathbf{G}^T \cdot \mathbf{b}]$ belong to the sphere. Then the vector $\mathbf{b}$ from that set which minimizes (4) is the solution to the joint ML detection and decoding problem.

We start by defining the set of the conditions that the components of a vector $\mathbf{b}$ need to satisfy so that $[\mathbf{G}^T \cdot \mathbf{b}]$ belongs to the searching sphere. To this end, we perform some pre-processing of the matrix $\mathbf{G}^T$.

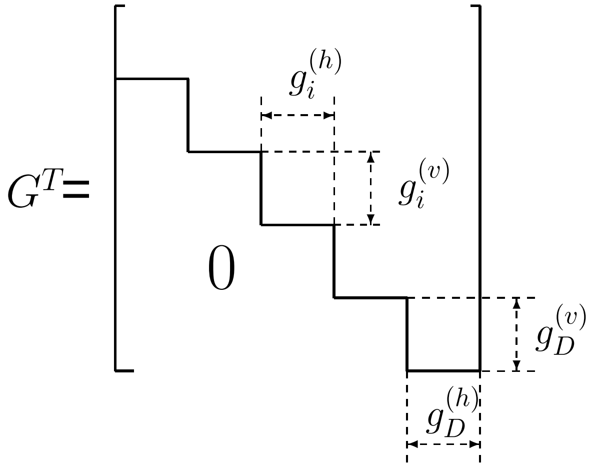

Figure 3: *Block upper-triangular form of* G^T

In particular, we transform the given matrix $\mathbf{G}^T$ into a block upper-triangular form, that is, starting from $\mathbf{G}$, we find its equivalent generator matrix G of the form shown in Figure 3. Note that $g_j^{(h)}$ in Figure 3 denotes the cardinality of the set of columns with $\sum_{i=j+1}^{D} g_i^{(v)}$ fixed zero entries, $j = 1, 2, \ldots, D$, and that D denotes the number of such distinct sets. For instance, we note that the columns 1 to $g_1^{(h)}$ have $\sum_{i=2}^{D} g_i^{(v)}$ fixed zero entries, the columns $g_1^{(h)} + 1$ to $g_1^{(h)} + g_2^{(h)}$ have $\sum_{i=3}^{D} g_i^{(v)}$ fixed zero entries, etc. In general, columns $\left(\sum_{i=1}^{j} g_i^{(h)} + 1\right)$ to $\sum_{i=1}^{j+1} g_i^{(h)}$ have $\sum_{i=j+2}^{D} g_i^{(v)}$ fixed zero entries, $j = 0, \ldots, D-1$. Clearly, $\sum_{i=1}^{D} g_i^{(h)} = k, \quad \sum_{i=1}^{D} g_i^{(v)} = m.$

We assume that the transformation of $\mathbf{G}^T$ to the form in Figure 3 is performed by a greedy algorithm that first finds the largest possible $g_D^{(v)}$, fixes it, proceeds to find the largest possible $g_{D-1}^{(v)}$, and so on. As we will discuss shortly, such a construction of G^T is beneficial for the computational complexity of the JDD-ML algorithm. The details of the greedy algorithm are given in Appendix A.

We refer to the set of the ratios, $\{g_1^{(h)}/g_1^{(v)}, \ldots, g_D^{(h)}/g_D^{(v)}\}$ as the *diagonal profile* of the matrix G^T. When $g_j^{(h)}/g_j^{(v)} = k/m$, $j = 1, \ldots, D$, we say that the diagonal profile is *uniform*. The reason for introducing the notion of the profile is that by focusing first on G^T with uniform profiles, we can derive a simple version of the JDD-ML algorithm and gain valuable intuition that we shall find useful later when considering the general case.

3.1 A special case: rate $1/2$ code, G^T with a uniform diagonal profile

We start the description of the algorithm by considering the special case of a rather simple block code. Assume that the information vector $\mathbf{b}$ is encoded by the rate $R_c = 1/2$ binary code for which G^T has uniform diagonal profile. Then the entries c_{2i-1} and c_{2i} of the coded vector $\mathbf{c}$ can be expressed as a linear combination of the bits $(b_i,\ b_{i+1}\ \ldots\ b_k)$ only, i.e., $c_{2i-1} = \sum_{j=i}^{k} G^T(2i-1, j) \cdot b_j$, and $c_{2i} = \sum_{j=i}^{k} G^T(2i, j) \cdot b_j$, where all operations are modulo 2. Recall the assumption that the size of the Galois field, whose elements comprise both the uncoded and coded vectors, is as same as the size of the PAM constellation from which the elements of the transmitted symbol vector are chosen. Therefore, the symbols s_{2i-1} and s_{2i}, just like the coded bits c_{2i-1} and c_{2i}, depend only on the information bits $(b_i,\ b_{i+1}\ \ldots\ , b_k)$.

The point $\mathbf{s}$ lies in a sphere of radius r around $\mathbf{x}$ if and only if it holds that

$$r^2 \geq \|\mathbf{x} - H\mathbf{s}\|^2 = (\mathbf{s} - \hat{\mathbf{s}})^* H^* H (\mathbf{s} - \hat{\mathbf{s}}), \tag{5}$$

where $\hat{\mathbf{s}} = H^{\dagger}\mathbf{x}$ is the unconstrained least-squares estimate. Introducing the QR decomposition $H = QR$, we can write the condition (5) as

$$r^2 \ \geq \ \sum_{i=1}^{k} \left[r_{2i-1,2i-1}^2 (s_{2i-1} - \hat{s}_{2i-1} + \sum_{j=2i}^{2k} \frac{r_{2i-1,j}}{r_{2i-1,2i-1}} (s_j - \hat{s}_j))^2 + r_{2i,2i}^2 (s_{2i} - \hat{s}_{2i} + \sum_{j=2i+1}^{2k} \frac{r_{2i,j}}{r_{2i,2i}} (s_j - \hat{s}_j))^2 \right] \tag{6}$$

Expanding the summations on the right-hand side of the inequality (6) and considering only the last term in it (i.e., the term for $i = k$), we can state an obvious necessary condition for $\mathbf{s}$ to belong to the sphere,

$$r_{2k,2k}^2 (s_{2k} - \hat{s}_{2k}) + r_{2k-1,2k-1}^2 \left[s_{2k-1} - \hat{s}_{2k-1} + \frac{r_{2k-1,2k}}{r_{2k-1,2k-1}} (s_{2k} - \hat{s}_{2k}) \right]^2 \leq r^2. \tag{7}$$

The terms on the left-hand side of (7) comprise the part of the summation in (6) which only depends on b_k. Therefore, condition (7) need to be tested for every $b_k \in GF(L)$. When, for some b_k, the inequality (7) is satisfied, that particular b_k value is substituted for in (6). Now, the part of the expression on the right-hand side of (6) that only depends on b_k can be evaluated and is taken to the left-hand side of (6) to yield r'^2,

$$r'^2 = r^2 - r_{2k,2k}^2 (s_{2k} - \hat{s}_{2k}) - r_{2k-1,2k-1}^2 \left[s_{2k-1} - \hat{s}_{2k-1} + \frac{r_{2k-1,2k}}{r_{2k-1,2k-1}} (s_{2k} - \hat{s}_{2k}) \right]^2 .$$

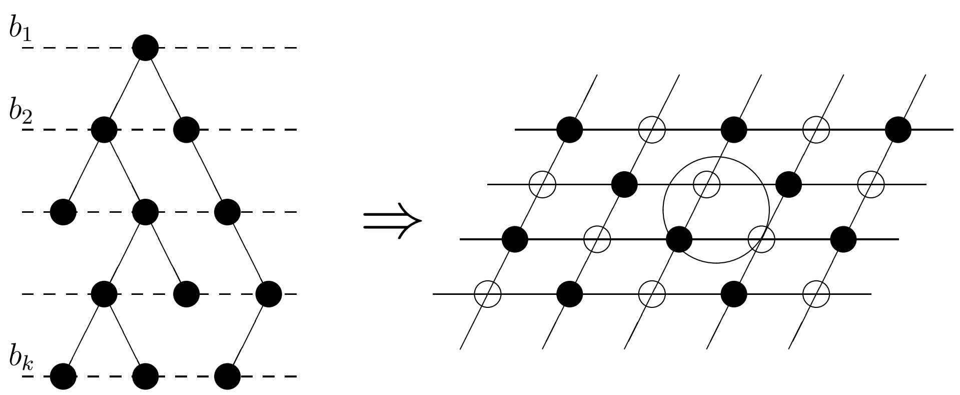

Figure 4: The tree-pruning interpretation of the JDD-ML algorithm

Then by considering the second to last term (i.e., the term for $i = k-1$) in the expanded summations (6), one can state a (stronger) necessary condition that b_{k-1} (assuming the already fixed value of b_k) needs to satisfy in order that the point $\mathbf{s}$ belongs to the sphere,

$$\begin{aligned} r'^2 \geq\ & r^2_{2k-3,2k-3}\big(s_{2k-3} - \hat{s}_{2k-3} + \textstyle\sum_{j=2k-2}^{2k} \frac{r_{2k-3,j}}{r_{2k-3,2k-3}}(s_j - \hat{s}_j)\big)^2 \\ & + r^2_{2k-2,2k-2}\big(s_{2k-2} - \hat{s}_{2k-2} + \textstyle\sum_{j=2k-1}^{2k} \frac{r_{2k-2,j}}{r_{2k-2,2k-2}}(s_j - \hat{s}_j)\big)^2. \end{aligned}$$

When such b_{k-1} is found, it is fixed and substituted for in (6). If no such b_{k-1} is found, we need to take one step back, discard the previously chosen higher-indexed bit (i.e., b_k), chose another one instead and proceed likewise. By continuing in the same way, we state the conditions on the remaining bits $(b_{k-2}, \ldots, b_1)$ and thus define the total of k nested necessary conditions from which all components of the vector $\mathbf{b}$ may consecutively be found.

We refer to the previously described procedure as the JDD-ML (joint ML detection and decoding) algorithm. One can think of the JDD-ML algorithm as a search on a tree (which, for the special case that we considered up to this point, is binary tree), as illustrated in Figure 4. The maximum depth of the tree is k. The conditions which, when violated, result in tree-pruning, are tested with respect to the integer lattice generated by H. Whenever a point falls outside the sphere centered at the received point $\mathbf{x}$ in the Euclidean space containing the lattice, the current node in the tree is discarded.

Each node at every level of the tree corresponds to a point in $GF(L)$. The paths on the tree that survive the pruning correspond to the information vectors which generate lattice points inside the sphere. The lattice points in the Euclidean space are related to the tree via both the (code generator) matrix G and the (lattice generating) channel matrix H. The block code maps the space with L^k elements (the information vector space) to the lattice with L^m points (the symbol space). [The "blank" points in Figure 4 denote lattice points that do not belong to the symbol space.]

The description of the JDD-ML algorithm based on (6) assumes a rate $R_c = 1/2$ binary block code with a uniform profile of G^T. The algorithm can be generalized to accommodate for an arbitrary diagonal profile of the arbitrary rate code generator matrix. To this end, we will find it useful to express the condition (6), still specialized for the $1/2$ rate code with uniform diagonal profile, in a matrix form. That is, we write it as

$$\sum_{j=1}^{k} \left\| R_{jj} \begin{pmatrix} s_{2j-1} - \hat{s}_{2j-1} \\ s_{2j} - \hat{s}_{2j} \end{pmatrix} + \sum_{i=j+1}^{k} R_{ji} \begin{pmatrix} s_{2i-1} - \hat{s}_{2i-1} \\ s_{2i} - \hat{s}_{2i} \end{pmatrix} \right\|^2 \leq r^2, \tag{8}$$

where $R_{jj} = R(2j-1:2j; 2j-1:2j)$, $R_{ji} = R(2j-1:2j; 2i-1:2i)$.

Expression (8) can now be used to state the set of conditions on $b_k, b_{k-1}, \ldots, b_1$, as we have done earlier in this section.

3.2 General case: arbitrary-rate codes, G^T with an arbitrary diagonal profile

To state the JDD-ML algorithm in a matrix form similar to (8) but for the general structure of G^T, it will be convenient to denote

$$m_i = \sum_{l=D-i+1}^{D} g_l^{(v)}, \text{ and } k_i = \sum_{l=D-i+1}^{D} g_l^{(h)}, \tag{9}$$

$i = 1, 2, \ldots, D$. In addition, define $m_0 = k_0 = 0$. The m_i and k_i, $i = 1, 2, \ldots D$ are illustrated in Figure 5.

Now, the condition (8) can be generalized for the case of G^T with an arbitrary diagonal profile as

$$\sum_{j=1}^{D} \| R_{jj} (\mathbf{s}_j - \hat{\mathbf{s}}_j) + \sum_{i=j+1}^{D} R_{ji} (\mathbf{s}_i - \hat{\mathbf{s}}_i) \|^2 \leq r^2, \tag{10}$$

where $R_{ji} = R(m_D - m_{D-j+1} + 1 : m_D - m_{D-j}; m_D - m_{D-i+1} + 1 : m_D - m_{D-i})$, and where $\mathbf{s}_j = \left[s_{m_D - m_{D-j+1}+1} \ \ldots \ s_{m_D - m_{D-j}} \right]^T$, $\hat{\mathbf{s}}_j = \left[\hat{s}_{m_D - m_{D-j+1}+1} \ \ldots \ \hat{s}_{m_D - m_{D-j}} \right]^T$, $j = 1, 2, \ldots, D$, $j \leq i \leq D$.

Upon careful inspection of (10), one can state a necessary condition that bits $(b_{k_D - k_1 + 1}, \ldots, b_k)$ need to satisfy in order for inequality (10) to hold,

$$\| R_{DD} (\mathbf{s}_D - \hat{\mathbf{s}}_D) \|^2 \leq r^2. \tag{11}$$

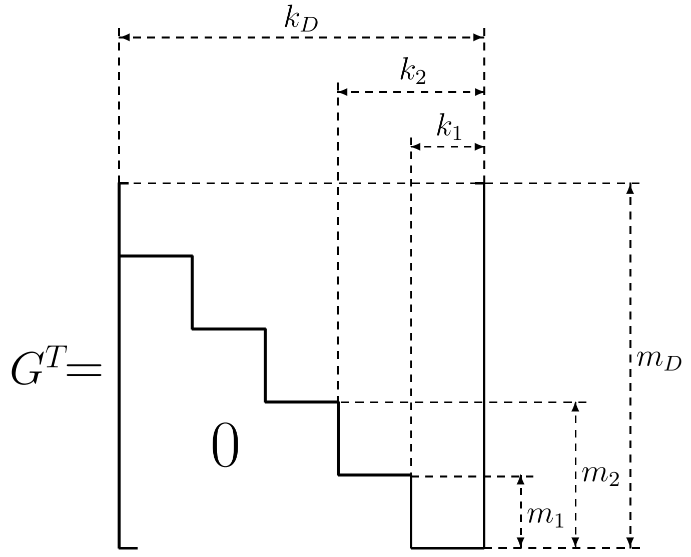

Figure 5: G^T *with an arbitrary diagonal profile.*

For every subvector $[b_{k_D-k_1+1}\ \dots\ b_k] \in GF(L)^{g_D^{(h)}}$ which satisfies condition (11), we go back to (10) and substitute in that particular $[b_{k_D-k_1+1}\ \dots\ b_k]$. Then a new necessary condition on $(b_{k_D-k_2+1}, \dots, b_{k_D-k_1})$ (and already chosen $(b_{k_D-k_1+1}, \dots, b_k)$) can be stated as

$$\sum_{j=1}^{D-1} \| R_{jj} \left(\mathbf{s}_j - \hat{\mathbf{s}}_j \right) + \sum_{i=j+1}^{D} R_{ji} \left(\mathbf{s}_i - \hat{\mathbf{s}}_i \right) \|^2 \leq r'^2,$$

where $r'^2 = r^2 - \| R_{DD}(\mathbf{s}_D(b_{k_D-k_1+1}, \dots, b_k) - \hat{\mathbf{s}}_D) \|^2$ is the updated radius.

The procedure is continued until all the components of the information vector $\mathbf{b}$ that satisfy (10) are found. If no vector $\mathbf{b}$ that satisfies (10) is found, the radius r is increased and the algorithm is restarted. On the other hand, in general, there may be more than one information vector $\mathbf{b}$ found by the algorithm. Then the one that minimizes (4) is the solution to the joint ML detection and decoding problem.

Now we can see the motivation for the previously described construction of G^T in Figure 5, where we first maximize $g_D^{(v)}$, then $g_{D-1}^{(v)}$, and so on. Clearly, with such a construction, the search tree is pruned faster – for instance, the larger the value of $g_D^{(v)}$, the more likely is condition (11) violated and the tree pruned early (i.e., it is pruned closer to the root).

Remark: There is an inherent trade-off between computational complexity of the decoding and the performance of the code. As indicated in this section, from the complexity standpoint, it is beneficial that the diagonal profile of the generator matrix be such that $g_D^{(v)}$ in Figure 3 is as large as possible, followed by

$g_{D-1}^{(v)}$ chosen as large as possible, and so on. However, it can easily be seen that the minimum distance of the code is upper-bounded by $g_1^{(v)}$. Therefore, the larger the minimum distance of the code, the higher the expected complexity of the JDD-ML algorithm applied for its decoding. $\square$

The JDD-ML algorithm can be summarized as follows:

1. *Input:* G, R, $\mathbf{x}$, $\hat{\mathbf{s}}$, r.

2. Set $i = D$, $r_m^2 = r^2$.

3. Set $B_i = -1$.

4. $B_i = B_i + 1$, $\mathbf{b}(m_D - m_{D-i+1} + 1 : m_D - m_{D-i}) = \text{dec2baseL}(B_i)^3$; if $B_i > L^{g_i^{(h)}}$, go to 9.

5. Calculate

$$\begin{array}{rcl} \mathbf{s}_i & = & \text{mod}\left[G^T(m_D - m_{D-i+1} + 1 : m_D - m_{D-i}, k_D - k_{D-i+1} + 1 : k_D)\right. \\ & \cdot & \left.\mathbf{b}(k_D - k_{D-i+1} + 1 : k_D), L\right] - \frac{L-1}{2} \cdot \mathbf{1}_{g_i^{(v)}}, \end{array}$$

   where $\mathbf{1}_j$ denotes an j-dimensional vector with all entries 1. Also, calculate

$$r_{i-1}^2 = r_i^2 - \|R_{i,i}(\mathbf{s}_i - \hat{\mathbf{s}}_i) + \sum_{j=i+1}^{D} R_{ij}(\mathbf{s}_j - \hat{\mathbf{s}}_j)\|^2.$$

6. (Feasibility test) If $r_{i-1}^2 < 0$, go to 4.

7. (Decrease i) Set $i = i - 1$.

8. If $i = 0$, solution found. Save $\mathbf{b}$ and go to 3.

9. (Increase i) $i = i + 1$; if $i = D + 1$, terminate algorithm, else go to 3.

3.3 An alternative algorithm useful for large-alphabet codes

In principle, the JDD-ML algorithm can be employed for joint detection and decoding of linear block codes over any field GF(L). As we described earlier in the section, the algorithm employs a branch-and-bound-like tree-search strategy, and at each level of the tree tests all nodes for satisfying a certain bound. Testing all

[3] dec2baseL($\cdot$) takes the argument in decimal systems and converts it to the base-L

nodes on a level does not present a challenge for binary codes but may, however, become computationally consuming for large L.

Ideally, we would prefer to limit the computations on each tree level to only those nodes which satisfy the aforementioned bound. [Note that this is what the basic sphere decoding algorithm does: it specifies intervals to which node coordinates must belong.] However, it is not obvious how to do that in the JDD-ML algorithm.

To this end, we propose a simple modification of the basic sphere decoding algorithm, which solves the ML joint detection and decoding problem and, for large-alphabet codes, may incur valuable savings over the JDD-ML algorithm. This is achieved in the following way: the sphere decoding algorithm is employed to solve

$$\min_{\mathbf{s}\in\mathcal{D}_L^m} \|\mathbf{x} - H\mathbf{s}\|^2,$$

and every time a point inside a sphere is found, we test whether it is a valid codeword or not (by using parity-check matrix or parity-check polynomial, depending on the type of the code; this is at most quadratic operation). Finally, the closest lattice point in the sphere which is a valid codeword is used to retrieve the original information vector $\mathbf{b}$.

The expected computational complexity of this scheme can be found by directly applying the results of [5]. In particular, it can be obtained by adding the complexity incurred by testing whether the (expected number of) lattice points are codewords or not to the expected complexity of the basic sphere decoding.

On the other hand, one can further improve the speed of this algorithm by implementing the Schnorr-Euchner strategy with radius update [12]. The only notable difference with respect to its counterpart used in the basic sphere decoding algorithm is that the radius is updated only if the currently considered lattice point is not only closer to the center of the sphere than any other previously found point, but is also a valid codeword.

4 The Computational Complexity of JDD-ML

As described in the previous section, the JDD-ML algorithm performs the tree search illustrated in Figure 4. Each node in the tree corresponds to a lattice point, and if that lattice point is outside a sphere, the tree is pruned. Therefore, the computational complexity of the JDD-ML algorithm is proportional to the number of lattice points that the algorithm visits. Clearly, the number of visited points depends on the choice of the radius: a smaller radius means that the condition (6) is more strict and, therefore, more tree nodes are pruned. Of course, the radius still needs to be large enough so that the algorithm finds at least one symbol point inside the sphere. As in [5], we choose the radius of the sphere according to the statistics of the noise. Clearly,

$$\frac{1}{2\sigma^2} \cdot \|\mathbf{v}\|^2 = \frac{1}{2\sigma^2} \cdot \|\mathbf{x} - H\mathbf{s}\|^2$$

is a χ^2 random variable with n degrees of freedom. Therefore, we may choose the radius to be the scaled variance of the noise, $r^2 = \alpha n \sigma^2$, in such a way that with a high probability there is a lattice point inside the sphere,

$$\int_0^{\alpha n/2} \frac{\lambda^{n/2-1}}{\Gamma(n/2)} e^{-\lambda} d\lambda = 1 - \epsilon,$$

where $1 - \epsilon$ is set to a value close to 1, say, $1 - \epsilon = 0.99$. If the point is not found, we can increase the probability $1 - \epsilon$, adjust the radius, and search again.

The number of the symbol points that lie inside the sphere depends on the particular instantiation of both the channel matrix H and the noise vector $\mathbf{v}$. Since they vary from one channel use to another, the complexity of the algorithm is a random variable. A way to characterize it is by means of its moments. In this section, we derive the closed form analytical expression for the expected complexity of the JDD-ML algorithm for binary block codes.

Expected complexity of the algorithm is proportional to the expected number of the symbol points inside the sphere. In particular, we can write

$$\mathcal{C}_{JDD-ML} = \sum_{i=1}^{D} E(\#\text{ of symbols in } m_i\text{-dimensional sphere of radius } r) \cdot f_p(m_i) \tag{12}$$

where m_i defined in (9) is the dimension in Euclidean space that corresponds to the i^{th} level in the tree, and where $f_p(m_i) = \left[2g^{(v)}_{D-i+1} m_i + g^{(v)}_{D-i+1}(2k_i - 2g^{(v)}_{D-i} + g^{(v)}_{D-i+1} + 1)\right] L^{g^{(h)}_{D-i+1}}$ denotes the number of operations (multiplications and additions) that the algorithm (in our implementation) performs per each visited point in the dimension m_i.

The complexity expression (12) reflects both the structure of the generator matrix in Figure 3 and the nature of the JDD-ML algorithm. Namely, as described in the previous section, the algorithm descends down the tree in such a way that at each step i it imposes a constraint on the subset of k_i information vector components, where k_i is defined in (9). In the corresponding Euclidean space (see Figure 4), the i^{th} step in this descent corresponds to an increment of the dimension of the space (in which we are searching for the lattice points inside the sphere of radius r) from m_{i-1} to m_i, because $m_i = m_{i-1} + g^{(v)}_{D-i+1}$.

To calculate the expected number of symbol points inside the sphere, we employ the technique first used in [5]. In particular, we start by posing the following question:

Assume that $\mathbf{s}_t \in \mathcal{D}_L^m$ is transmitted and that $\mathbf{x} = H\mathbf{s}_t + \mathbf{v}$ is received; what is the probability that an arbitrary lattice point belongs to the sphere of radius r centered at $\mathbf{x}$?

The event which we need to probabilistically characterize is illustrated in Figure 6.

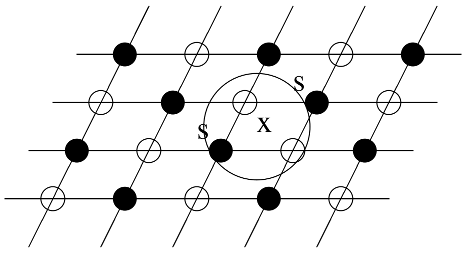

Figure 6: *The sphere $\mathcal{S}_t$ is centered at $\mathbf{x} = H\mathbf{s}_t + \mathbf{v}$; we are interested in finding the probability that $H\mathbf{s}_a \in \mathcal{S}_t$.*

Since the JDD-ML algorithm performs the search by going through the dimensions $m_i, i = 1, 2, \dots, D$, we need to calculate the previously mentioned probability for each dimension m_i. Results obtained in [5] imply that this probability is given by

$$\gamma\left(\frac{r^2}{2(\sigma^2 + \|\mathbf{s}_a^{m_i} - \mathbf{s}_t^{m_i}\|^2)}, \frac{m_i}{2}\right) = \int_0^u \frac{\lambda^{m_i/2-1}}{\Gamma(m_i/2)} e^{-\lambda} d\lambda, \tag{13}$$

where the upper limit of the integration on the right-hand side is given by $u = \frac{r^2}{2(\sigma^2 + \|\mathbf{s}_a^{m_i} - \mathbf{s}_t^{m_i}\|^2)}$, where $\mathbf{s}_t^{m_i}$ and $\mathbf{s}_a^{m_i}$ denote m_i-dimensional transmitted and arbitrary symbol vectors, respectively, and where $\gamma(a, b)$ denotes the incomplete gamma function of argument a and b degrees of freedom.

Given the probability (13), the expected complexity in (12) can be evaluated by going over all the points $\mathbf{s}_a^{m_i}$ in m_i-dimensional subspace of the symbol space $\mathcal{D}_L^m$ and summing up the corresponding probabilities (13), for all dimensions m_i. The result, averaged over the choice of $\mathbf{s}_t$, yields the desired expected number of points, and is given by

$$\sum_{i=1}^{D} \frac{1}{L^{m_i}} \sum_{\mathbf{s}_t^{m_i}, \mathbf{s}_a^{m_i}} \gamma\left(\frac{r^2}{2(\sigma^2 + \|\mathbf{s}_a^{m_i} - \mathbf{s}_t^{m_i}\|^2)}, \frac{m_i}{2}\right). \tag{14}$$

Using (14) we can, in principle, always calculate the expected complexity. However, although this calculation is to be done off-line, going over all pairs of points $(\mathbf{s}_t^{m_i}, \mathbf{s}_a^{m_i})$ may be quite time consuming.

Hence we search for ways to ease the calculation of the expression (14) by enumerating the information vector space. Recall that the transmitted vector $\mathbf{s}_t$ and the arbitrary vector $\mathbf{s}_a$ are obtained by encoding and modulating information vectors $\mathbf{b}_t$ and $\mathbf{b}_a$, respectively, i.e., $\mathbf{s}_t = [G^T\mathbf{b}_t]$, $\mathbf{s}_a = [G^T\mathbf{b}_a]$. The m_i-dimensional vectors $\mathbf{s}_t^{m_i}$ and $\mathbf{s}_a^{m_i}$ needed for the enumeration are given by $\mathbf{s}_t^{m_i} = [G_i^T\mathbf{b}_t^{k_i}]$ and $\mathbf{s}_a^{m_i} = [G_i^T\mathbf{b}_a^{k_i}]$, where G_i^T denotes $m_i \times k_i$ lower-right submatrix of G^T, and where $\mathbf{b}_t^{k_i}$ and $\mathbf{b}_a^{k_i}$ are k_i-dimensional information vectors. Therefore, if we can efficiently count the number of vectors $\mathbf{b}_a^{k_i}$ which, for given $\mathbf{b}_t^{k_i}$, give the same probability in (13), we can significantly speed-up the calculation of the expected number of points.

Now, note that the probability (14) is only a function of the Euclidean distance between $\mathbf{s}_t^{m_i}$ and $\mathbf{s}_a^{m_i}$, and thus we can write (14) as

$$\sum_{i=1}^{D} \frac{1}{L^{m_i}} \sum_{\|\mathbf{s}_a^{m_i} - s_t^{m_i}\|^2 = l} \gamma\left(\frac{r^2}{2(\sigma^2+l)}, \frac{m_i}{2}\right). \tag{15}$$

Therefore, we need to count the number of pairs ($\mathbf{b}_t^{k_i}$,$\mathbf{b}_a^{k_i}$) which generate equidistant pairs ($\mathbf{s}_a^{m_i}, \mathbf{s}_t^{m_i}$), i.e., count the number of pairs ($\mathbf{b}_t^{k_i}$,$\mathbf{b}_a^{k_i}$) such that for each integer $l > 0$,

$$\|\mathbf{s}_a^{m_i} - \mathbf{s}_t^{m_i}\|^2 = \left\|[G_i^T \cdot \mathbf{b}_a^{k_i}] - [G_i^T \cdot \mathbf{b}_t^{k_i}]\right\|^2 = l. \tag{16}$$

We demonstrate the enumeration procedure for $L = 2$, i.e., perform the counting in (16) for the case of the binary block codes. We start by making the following observations in relation to (16):

- Clearly, if $\mathbf{b}_a^{k_i} = \mathbf{b}_t^{k_i}$, then $\|\mathbf{s}_a^{m_i} - \mathbf{s}_t^{m_i}\|^2 = 0$.
- Assume that $\mathbf{b}_a^{k_i}$ differs from $\mathbf{b}_t^{k_i}$ in only entry, and note that

$$\|\mathbf{s}_a^{m_i} - \mathbf{s}_t^{m_i}\|^2 = (s_{a,1}^{m_i} - s_{t,1}^{m_i})^2 + \ldots + (s_{a,m_i}^{m_i} - s_{t,m_i}^{m_i})^2. \tag{17}$$

  Then if $\mathbf{b}_t^{k_i}$ and $\mathbf{b}_a^{k_i}$ have the j^{th} bit different, there will be w_j^i non-zero terms in the sum on the right-hand side of (17), where w_j^i, $j = 1, \ldots, k$, denotes the weight of the j^{th} column of G_i^T, i.e., w_j^i denotes the sum of all entries in that column. Let Y_1^i denote the set of the distinct column weights of G_i^T, and let the elements of the set y_1^i count the multiplicity of its weights, i.e., $y_1^i(l)$ is the number of columns of G_i^T whose weight is $Y_1^i(l)$. Clearly, $\|\mathbf{s}_a^{m_i} - \mathbf{s}_t^{m_i}\|^2 \in Y_1^i$.

- Assume that $\mathbf{b}_a^{k_i}$ differs from $\mathbf{b}_t^{k_i}$ in two positions. Then to enumerate all the possible values of

$\|\mathbf{s}_a^{m_i} - \mathbf{s}_t^{m_i}\|^2$, one needs to consider exclusive-or (XOR) sums of any two columns of G_i^T. Let Y_2^i denote the set of the distinct weights of XOR sums of any two columns of G_i^T, and let y_2^i denote the set counting the multiplicity of those weights, i.e., $y_2^i(l)$ is the number of pairs of columns of G_i^T whose XOR sum has weight $Y_2^i(l)$. It follows that $\|\mathbf{s}_a^{m_i} - \mathbf{s}_t^{m_i}\|^2 \in Y_2^i$.

- We proceed alike for the cases when $\mathbf{b}_a^{k_i}$ differs from $\mathbf{b}_t^{k_i}$ in more than two entries. In general, if $\mathbf{b}_a^{k_i}$ and $\mathbf{b}_t^{k_i}$ differ in j entries, we consider set of the vectors obtained by taking XOR sums of all possible combinations of j columns of G_i^T. Collect the distinct weights of such vectors into Y_j^i, and denote the set of the corresponding multiplicities by y_j^i. Then, $\|\mathbf{s}_a^{m_i} - \mathbf{s}_t^{m_i}\|^2 \in Y_j^i$.

In addition, we define $Y_0^i = \{0\}$, $y_0^i = \{1\}$, and note that

$$\sum_{l=1}^{|Y_j^i|} y_j^i(l) = \left(\begin{array}{c} k_i \\ j \end{array} \right), \quad j = 0, 1, \ldots, k_i.$$

Combining (12), (15), and the previously described enumeration, we obtain the following theorem.

Theorem 1. [Expected complexity of the JDD-ML algorithm for binary codes and fixed G]

Consider the model

$$\mathbf{x} = H\mathbf{s} + \mathbf{v},$$

where $\mathbf{v} \in \mathcal{R}^{n\times 1}$ is comprised of i.i.d. $\mathcal{N}(0,1)$ entries, $H \in \mathcal{R}^{n\times m}$ is comprised of i.i.d. $\mathcal{N}(0,\rho/m)$ entries, and $\mathbf{s} \in \mathcal{D}_2^m$ is an m-dimensional vector whose entries are obtained by modulating coded vector $\mathbf{c}$ onto a 2-PAM constellation. Furthermore, the vector $\mathbf{c}$ is obtained by block coding information vector $\mathbf{b}$,

$$\mathbf{c} = G^T \cdot \mathbf{b},$$

where $G^T \in GF(2)^{m\times k}$ has a diagonal profile $\{g_1^{(h)}/g_1^{(v)}, \ldots, g_D^{(h)}/g_D^{(v)}\}$. Then the expected complexity of the JDD-ML algorithm for the integer least-squares problem

$$\min_{\mathbf{b}\in GF(L)^k} \|\mathbf{x} - H\mathbf{s}\|^2,$$

for the given G, averaged over H and $\mathbf{v}$, is

$$C_{JDD-ML} = \sum_{i=1}^{D} f_p(m_i) \sum_{j=1}^{k_i} \sum_{l=0}^{|Y_j^i|} y_j^i(l) \gamma \left(\frac{r^2}{2(\sigma^2 + Y_j^i(l))}, \frac{m_i}{2} \right), \tag{18}$$

where Y_j^i denotes the set of the distinct weights of XOR sums of any j columns of G_i^T, $|Y_j^i|$ is the cardinality of Y_j^i, and where y_j^i counts the multiplicity of the weights in Y_j^i. Furthermore, G_i^T denotes the $m_i \times k_i$ lower-right submatrix of G^T, $m_i = \sum_{l=D-i+1}^{D} g_l^{(v)}$, and $k_i = \sum_{l=D-i+1}^{D} g_l^{(h)}$, $i = 1, 2, \ldots, D$.

Proof:

The proof follows from the previous discussion in this section.

□

Theorem 1 gives the expected complexity of the algorithm for a fixed G. The expected complexity of the algorithm for random G may also be of interest.[4] To this end, one should find the number of the tree points visited by the algorithm, averaged over H, $\mathbf{v}$, and G. In other words, starting from (15), we should find

$$E_{|G}\left\{\sum_{i=1}^{D} \frac{1}{L^{m_i}} \sum_{l} \gamma\left(\frac{r^2}{2(\sigma^2+l)}, \frac{m_i}{2}\right) N_l^{m_i}\right\},$$

where $N_l^{m_i}$ denotes the number of pairs $(\mathbf{s}_a^{m_i}, s_t^{m_i})$ such that $\|\mathbf{s}_a^{m_i} - s_t^{m_i}\|^2 = l$. This computation is rather involved and we will not attempt it here.

5 The JDD-ML algorithm for Cyclic Codes

As shown in the previous sections, the JDD-ML algorithm can be employed for joint detection and decoding of unstructured (e.g., random) linear block codes. All that the algorithm requires is some pre-processing of the generator matrix. Of course, certain code structures may be preferred from the perspective of the algorithm complexity.

In coding theory, however, practical constraints have led researchers to develop linear codes that are highly structured and allow for efficient implementation of both encoder and decoder. In this section, we focus on the so-called *cyclic codes* (see, e.g., [11]), which include, e.g., BCH and Reed-Solomon codes. These codes allow for efficient encoding by means of shift-register encoder structure. Furthermore, there are efficient syndrome polynomial based techniques [11] for decoding of the cyclic codes on scalar channels, as well as low-complexity soft-decision techniques which provide excellent performances on scalar channels [13, 14]. We will not dwell on those here. Instead, as in the rest of this paper, we will focus on the transmission over the (Gaussian) vector channel.

An (m, k) cyclic code is defined via its generating polynomial, $g(x) = g_1 + g_2 x + \ldots + g_l x^{l-1}$, where

[4]For instance, the results for random G should also be indicative of the complexity behavior of large, unstructured generator matrices $G^T \in GF(2)^{m \times k}$ with the same diagonal profile.

$l = m - k + 1,\ g_i \in GF(L)$. The corresponding generator matrix G is given by

$$G^T = \begin{bmatrix} g_1 & & & \\ g_2 & g_1 & & \\ \vdots & \ddots & \ddots & \\ g_l & \ddots & \ddots & g_1 \\ & g_l & \ddots & g_2 \\ & & \ddots & \vdots \\ & & & g_l \end{bmatrix}. \tag{19}$$

The generic JDD-ML algorithm studied in the previous sections may now be directly employed for joint detection and decoding in the systems employing the cyclic codes with G^T in (19). However, by consulting Figure 3, we note that no pre-processing of the matrix G^T in (19) is required since it already has (some) upper-triangular structure. The special Toeplitz structure of G^T simplifies the algorithm and we state it here.

Note that we can now write the condition (5) as

$$\begin{aligned} r^2 &\geq (s - \hat{s})^* R^* R (s - \hat{s}) \\ &= \underbrace{\sum_{i=m-k+1}^{m} r_{i,i}^2 \left(s_i - \hat{s}_i + \sum_{j=i+1}^{m} \frac{r_{i,j}}{r_{i,i}} (s_j - \hat{s}_j) \right)^2}_{=S_1} + \underbrace{\sum_{i=1}^{m-k} r_{i,i}^2 \left(s_i - \hat{s}_i + \sum_{j=i+1}^{m} \frac{r_{i,j}}{r_{i,i}} (s_j - \hat{s}_j) \right)^2}_{=S_2} \\ &= S_1 + S_2. \end{aligned} \tag{20}$$

The $k-1$ terms in the expansion of S_1 on the right-hand side of the inequality (20) can be used to state the conditions for finding $b_2, \ldots, b_k$. Note that the symbol components $s_{m-k+1}, \ldots, s_m$ which appear in the expression for S_1 can be found as

$$s_{m-k+j} = \sum_{q=1}^{\min(l,k-j+1)} g_{l-q+1} \cdot b_{m-k+j+q-1}, \quad j = 1, \ldots, k.$$

Now, considering the last term in the expansion of S_1, we can state an obvious necessary condition for $\mathbf{s}$ to belong to the sphere,

$$r_{m,m}^2 (s_m - \hat{s}_m)^2 \leq r^2. \tag{21}$$

The expression on the left-hand side of (21) only depends on b_k. Therefore, we can test (21) for every

$b_k \in GF(L)$. When, for some b_k, the inequality (21) is satisfied, that particular b_k value (and, therefore, s_m) is substituted for in (20). Now, the part of the expression on the right-hand side of (20) that only depends on b_k can be evaluated and is taken to the left-hand side of (6) to yield $r'^2 = r^2 - r^2_{m,m}(s_m - \hat{s}_m)^2$.

Then, by considering the next term in S_1, one can state a (stronger) necessary condition that b_{k-1} (assuming the already fixed value of b_k) needs to satisfy in order that the point $\mathbf{s}$ belongs to the sphere,

$$r^2_{m-1,m-1}\left(s_{m-1} - \hat{s}_{m-1} + \frac{r_{m-1,m}}{r_{m-1,m-1}}(s_m - \hat{s}_m)\right)^2 \quad \leq \quad r'^2.$$

When such b_{k-1} is found, it is fixed and substituted for in (20). If no such b_{k-1} is found, we need to take one step back, discard the previously chosen higher-indexed bit (i.e., b_k), chose another one instead and proceed likewise. By continuing in the same way, we state the conditions on the bits $(b_{k-2}, \ldots, b_2)$.

Having determined $b_k, \ldots, b_2$ which satisfy $S_1 < r^2$, we use S_2 to find b_1. In particular, b_1 must be such that $S_2 \leq r^2 - S_1$, where the symbols $s_1, \ldots, s_{m-k}$ are determined as

$$s_j = \sum_{q=1}^{\min(l,j)} g_q \cdot b_{j-q+1}, \quad j = 1, \ldots, m-k.$$

If such b_1 exist, then $\mathbf{b} = [b_1 \;\; b_2 \;\; \ldots \;\; b_k]$ is the solution to (20). If no such b_1 is found, the algorithm takes a step back up the tree, chooses another b_{k-1}, and proceeds.

The JDD-ML algorithm for cyclic codes can be summarized as follows:

1. *Input:* G, R, $\mathbf{x}$, $\hat{\mathbf{s}}$, r.

2. Set $i = k$, $r^2_m = r^2$, $\hat{s}_{m|m+1} = \hat{s}_m$.

3. Set $b_i = -1$.

4. $b_i = b_i + 1$; if $b_i > L$, go to 9.

5. If $i > 1$, calculate

$$s_{m-k+i} = \sum_{q=1}^{\min(l,k-i+1)} g_{l-q+1} \cdot b_{m-k+i+q-1},$$

$$\hat{s}_{m-k+i-1|m-k+i} = \hat{s}_{m-k+i-1} - \sum_{j=m-k+i}^{m} \frac{r_{m-k+i-1,j}}{r_{m-k+i-1,m-k+i-1}}(s_j - \hat{s}_j),$$

$$r^2_{m-k+i-1} = r^2_{m-k+i} - r^2_{m-k+i,m-k+i}(s_{m-k+i} - \hat{s}_{m-k+i|m-k+i+1})^2.$$

Otherwise, if $i = 1$, for $j = m-k, m-k-1, \ldots, 1$ calculate

$$s_j = \sum_{q=1}^{\min(l,j)} g_q \cdot b_{j-q+1}, \quad \hat{s}_{j|j+1} = \hat{s}_j - \sum_{q=j+1}^{m} \frac{r_{j,q}}{r_{j,j}}(s_q - \hat{s}_q), \quad r_j^2 = r_{j+1}^2 - r_{j,j}^2(s_j - \hat{s}_{j|j+1})^2.$$

6. (Feasibility test) If ($i > 1, r_{m-k+i}^2 < 0$) or ($i = 1, r_1^2 < 0$), go to 4.

7. (Decrease i) Set $i = i - 1$.

8. If $i = 0$, solution found. Save $\mathbf{b}$ and go to 3.

9. (Increase i) $i = i + 1$; if $i = k + 1$, terminate algorithm, else go to 3.

We note that the algorithm is better suited for high-rate codes. In particular, for the high-rate codes, $l = m - k$ is relatively small in comparison with m, which is beneficial from the complexity standpoint because the sphere radius r is linear function of m. If l were not small in comparison with m, as when the rate of the code is very low, the conditions from which we find $b_k, \ldots, b_2$ would be too loose and there would be not much pruning in the tree.

6 The JDD-MAP Algorithm and its Complexity

The joint ML detection and decoding problem (4) assumes no prior knowledge about the information vector $\mathbf{b}$. There are scenarios, however, when we may have the access to the a priori information, that is, when we know the set of the a priori probabilities $\{p(b_1), p(b_2), \ldots, p(b_k)\}$. This a priori information may be exploited in order to obtain the maximum a posteriori estimate of the information vector $\mathbf{b}$. The joint maximum a posteriori detection and decoding algorithm (JDD-MAP) solves the optimization problem

$$\max_{\mathbf{b} \in GF(L)^k} p(\mathbf{b}|\mathbf{x}),$$

which can be expressed as arg $\max_{\mathbf{b} \in GF(L)^k} p(\mathbf{b}|\mathbf{x}) =$ arg $\max_{\mathbf{b} \in GF(L)^k} p(\mathbf{x}|\mathbf{b})p(\mathbf{b})$. Assuming independent bits $b_1, \ldots, b_k$, we can write $p(\mathbf{b}) = \prod_{i=1}^{k} p(b_i) = e^{\sum_{i=1}^{k} \log p(b_i)}$, and the joint MAP detection and decoding problem may be stated as

$$\min_{\mathbf{b} \in GF(L)^k} \left[\left\| \mathbf{x} - H[G^T \mathbf{b}] \right\|^2 - \sum_{i=1}^{k} \log p(b_i) \right]. \tag{22}$$

To solve (22), we take the same approach as we did for the joint ML detection and decoding problem. In particular, we search for vectors $\mathbf{b}$ such that

$$\sum_{j=1}^{D}\left\|R_{jj}\left(\mathbf{s}_j-\hat{\mathbf{s}}_j\right)+\sum_{i=j+1}^{D}R_{ji}\left(\mathbf{s}_i-\hat{\mathbf{s}}_i\right)\right\|^2\le r^2+\sum_{i=1}^{k}\log p(b_i), \tag{23}$$

where R_{ji}, $\mathbf{s}_j$, and $\hat{\mathbf{s}}_j$ are defined in Section 3.2.

From (23) it is clear that we search for the lattice points that no longer belong to the sphere but rather lie in some distorted object in the Euclidean space. This object can be thought of as the sphere stretched or compressed in various dimensions, depending on the a priori confidence that we have about the corresponding components of the vector $\mathbf{b}$.

From (23), one can state a necessary condition that bits $(b_{k_D-k_1+1},\ldots,b_k)$ need to satisfy in order for inequality (23) to hold,

$$\left\|R_{DD}(\mathbf{s}_D-\hat{\mathbf{s}}_D)\right\|^2\le r^2+\sum_{j=k_D-k_1+1}^{k}\log p(b_j). \tag{24}$$

For every subvector $[b_{k_D-k_1+1}\ \ldots\ b_k]\in GF(L)^{g_D^{(h)}}$ which satisfies condition (24), we go back to (23) and substitute in that particular $[b_{k_D-k_1+1}\ \ldots\ b_k]$. Then a new, more strict necessary condition on $(b_{k_D-k_2+1},\ldots,b_{k_D-k_1})$ and already chosen $(b_{k_D-k_1+1},\ldots,b_k)$ is stated as

$$\sum_{j=1}^{D-1}\left\|R_{jj}\left(\mathbf{s}_j-\hat{\mathbf{s}}_j\right)+\sum_{i=j+1}^{D}R_{ji}\left(\mathbf{s}_i-\hat{\mathbf{s}}_i\right)\right\|^2\le r'^2+\sum_{j=k_D-k_2+1}^{k_D-k_1}\log p(b_j),$$

where $r'^2=r^2-\left\|R_{DD}(\mathbf{s}_D(b_{k_D-k_1+1},\ldots,b_k)-\hat{\mathbf{s}}_D)\right\|^2-\sum_{j=k_D-k_1+1}^{k}\log p(b_j)$ is the updated radius. The procedure is continued until all the components of the information vector $\mathbf{b}$ that satisfy (23) are found. If no vector $\mathbf{b}$ that satisfies (23) is found, the radius r is increased and the algorithm is restarted. On the other hand, in general, there may be more than one information vector $\mathbf{b}$ found by the algorithm. Then the one that minimizes (22) is the solution to the joint MAP detection and decoding problem.

The computational complexity of the JDD-MAP algorithm appears difficult to compute in closed form. However, we can bound its complexity by relating it to the complexity of the JDD-ML algorithm. In particular, the probability that an arbitrary information vector $\mathbf{b}_a$ generates a lattice point $\mathbf{s}_a$ inside the sphere around $H\mathbf{s}_t$, where $\mathbf{s}_t$ is the transmitted symbol (generated by $\mathbf{b}_t$), is given by

$$p_{\mathbf{b}_a} = \int_0^{\frac{r^2+\sum_{i=1}^k \log p(b_i)}{2(\sigma^2+\|s_a-s_t\|^2)}} \frac{\lambda^{m/2-1}}{\Gamma(m/2)} e^{-\lambda} d\lambda.$$

Since $\sum_{j=1}^k \log p(b_j) \leq 0$, we have

$$\frac{r^2+\sum_{i=1}^k \log p(b_i)}{2(\sigma^2+\|s_a-s_t\|^2)} \leq \frac{r^2}{2(\sigma^2+\|s_a-s_t\|^2)}.$$

Since the incomplete gamma function is monotonically increasing with its argument, it follows that $p_{\mathbf{b}_a}^{JDD-MAP} \leq p_{\mathbf{b}_a}^{JDD-ML}$, and we conclude that, for the same choice of radius r, the number of lattice points visited by the JDD-MAP algorithm is upper bounded by the number of lattice points visited by the JDD-ML algorithm. Note, however, that there is a small increase in the number of operations per each visited point due to computations involving the a priori information.

The JDD-MAP algorithm is particularly promising for implementation in communication schemes employing concatenated coding (with a block inner code) and iterative decoding. In those applications, we generally choose the radius of the search r so to obtain good approximation of the soft information typically required by the receiver (see [10]). On another note, we should point out that, following [9], soft decisions may also be obtained by using the JDD-ML algorithm as a list decoder.

7 Performance Simulations

In this section, we study the bit-error rate (BER) performance and the expected computational complexity of the proposed algorithms in a few examples.

Example 1: We consider the rate $R = 1/2$, $(12, 24)$ binary random codes (i.e., $m = 24$, $k = 12$, $L = 2$). In addition, we consider the Golay 24 code and compare its performance and detection/decoding complexity with that of the random codes.

Figure 7 compares the performance of the JDD-ML algorithm with a two-stage detector/decoder which first detects the transmitted modulated codeword, $\mathbf{s}$, and then decodes the original information word, $\mathbf{b}$. [Note that, since there are no efficient alternative ML detectors, we use the standard sphere decoder in the first stage.] The code is randomly chosen from a collection of codebooks that is available at both the transmitter and the receiver; the particular choice of the codebook is also known to both the transmitter and

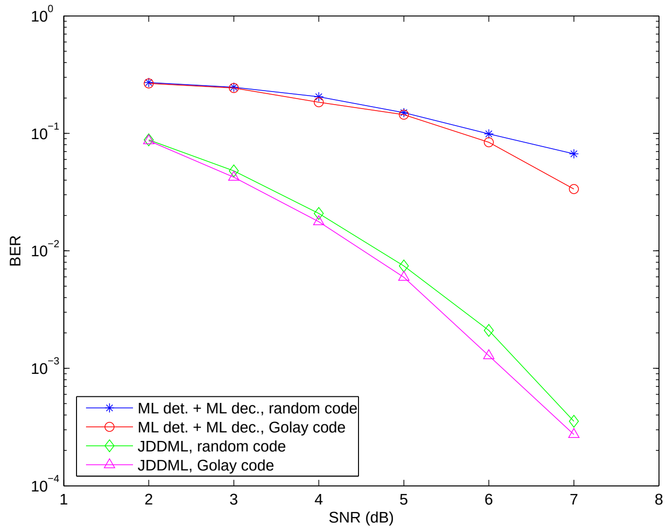

Figure 7: *BER performance of the JDD-ML algorithm employed for joint detection and decoding of the random codes and the Golay code, compared with the performance of the two-stage ML detector/decoder.*

the receiver. The BER performance is averaged over many realizations of random codes. Clearly, the JDD-ML algorithms significantly outperforms the two-stage detection/decoding algorithm. Furthermore, we note that Golay code outperforms the random code; this is expected as the Golay code has the best minimum distance properties among all codes of dimension $m = 24$.

In Figure 8, we show the expected complexity exponent of the JDD-ML algorithm for decoding random codes. The complexity exponent is defined as $e = \log_m C$, where C represents the total flop count in (18). In the considered range of SNR, the expected complexity exponent of the JDD-ML algorithm for joint detection and decoding of random codes is ≤ 4. In the same figure, we plot the expected complexity exponent of the JDD-ML algorithm for the Golay code, which is slightly higher than that the one for the random code. This illustrates the discussion on the trade-of between performance and complexity in Section 3: the Golay code has the best minimum distance property and thus its generator matrix imposes greater computational burden on decoding than the average random code does. Finally, for a comparison, we include the expected complexity exponent of exhaustive search, which is much greater than that of the JDD-ML algorithm.

In Figure 9, we compare the BER performance of the JDD-ML algorithm with that of the 2-stage decoder consisting of the list sphere decoder followed by the soft-decoding algorithm based on order statistics decoding (OSD, order-5) proposed in [14]. Note that the list sphere decoder generates soft information based on all points inside a sphere (and not only those that are valid codewords), which deteriorates the overall

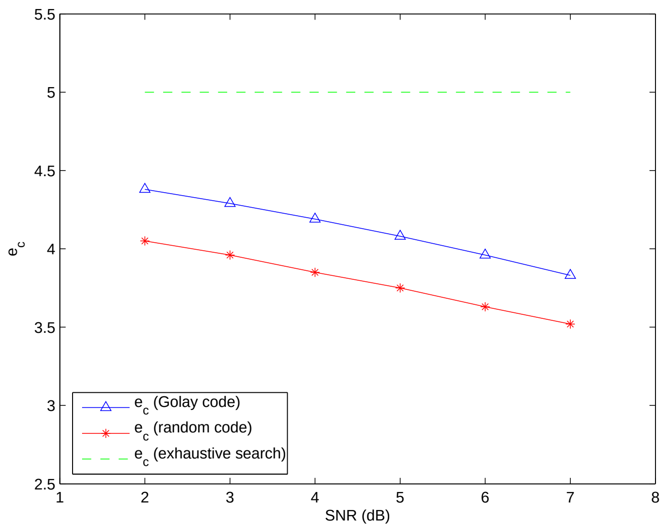

Figure 8: *The expected complexity exponent of the JDDML algorithm for the Golay and random codes, and the complexity exponent of exhaustive search.*

BER performance. This is, in fact, confirmed in Figure 9, where the BER performance of the JDD-ML algorithm is clearly much better than that of the considered soft-decision scheme.

Example 2: We consider the $(15, 11)$ Reed-Solomon code, $n = m = 16$, and study the BER performance of the joint ML detection and decoding algorithm proposed in Section 3.3. The algorithm employs the Schnorr-Euchner search strategy with radius update and, as shown in Figure 10, significantly outperforms the 2-stage receiver employing hard ML detection followed by the ML decoding. On the other hand, as shown in Figure 11, the complexity exponent of the algorithm is much smaller than that of the exhaustive search. In fact, comparing Figure 10 and Figure 11, in the SNR regime where the BER performance of the algorithm is roughly around 10^{-4}, its expected complexity exponent is ≈ 4.

Example 3: Finally, we consider the setup of Example 1 but employ a concatenated coding scheme with an outer convolutional code and the inner Golay code. The information bit sequence with 504 bits is encoded by a rate $R = 1/2$ convolutional code with memory length 2 and generating polynomials $G_1(D) = 1 + D^2$ (feedforward) and $G_2(D) = 1 + D + D^2$ (feedback). The coded sequence is then further encoded by the Golay code, mapped onto a 2-PAM modulation scheme, and transmitted across a Gaussian channel H [Note that H is a 24×24 matrix and, in each channel use, we may transmit 24 bits; to transmit the entire coded sequence of length $504 * 2 * 2 = 2016$, we need to use the channel 84 times.] On the receiver side, the JDD-MAP algorithm finds soft information for the inner (Golay) code, and passes them on to the soft decoder for

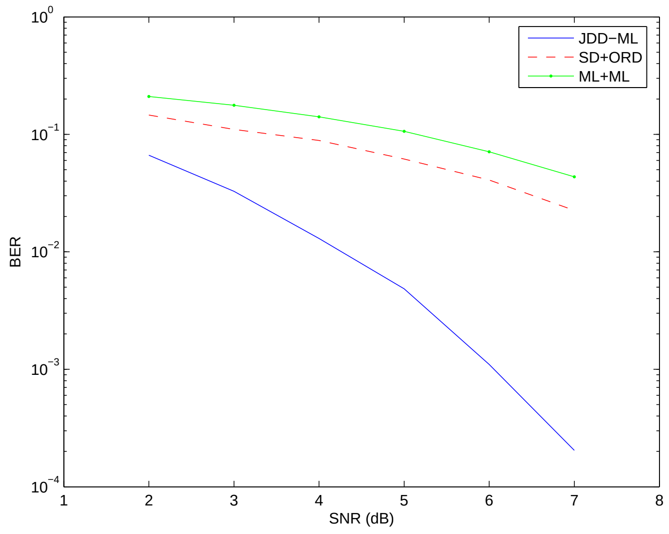

Figure 9: *BER performance of the JDD-ML algorithm compared with the BER performance of the ordered-statistics soft-decision decoding algorithm.*

the outer (convolutional) code. The two decoders iterate the soft information. As shown in Figure 12, the soft iterative decoding significantly outperforms the system employing hard (ML) decisions.

8 Discussion and Conclusion

In this paper, we considered the problem of joint detection and decoding for linear block codes on Gaussian vector channels. We focused on the maximum-likelihood and maximum a posteriori criteria for the design of the receiver. Due to the potentially rather high complexity of the joint solution, the design of the two receiver components, detector and decoder, are typically treated separately in practice. However, performance losses suffered by the systems employing heuristic solutions motivates the search for efficient algorithms that treat the problem of the receiver design jointly.

Drawing on the ideas encountered in solving standard integer least-squares problems (in particular, the sphere decoding algorithm), we developed algorithms that solve both the joint maximum-likelihood and joint maximum a posteriori detection and decoding problems. We proposed the JDD-ML algorithm which solves the joint ML detection and decoding by performing sphere-constrained search for the lattice points which are valid codewords. Due to the probabilistic setting of the problem, the computational complexity of the JDD-ML algorithm is a random variable. We quantified it by means of the expected complexity, which for the case of binary codes we found analytically, in a closed form. The expected complexity of the

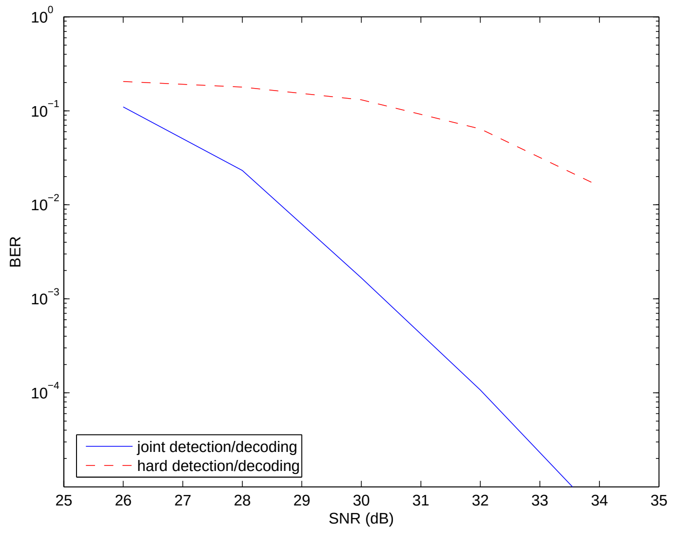

Figure 10: *The BER performance of the joint detection and decoding algorithm for the Reed-Solomon* $(15, 11)$ *code, and the BER performance of the receiver that performs (hard) ML detection followed by ML decoding.*

JDD-ML algorithm was, in examples, shown to be a low-degree polynomial in the length of the uncoded information word over the considered range of SNR. We also proposed an efficient alternative algorithm for large-alphabet codes. Furthermore, we considered the MAP joint detection and decoding problem for the case when the a priori information for the uncoded data are provided to the receiver. We derived the JDD-MAP algorithm for solving the above problem and studied its expected complexity. Simulations show that the soft decision scheme employing the JDD-MAP algorithm significantly outperforms scheme that uses hard decisions.

The algorithms presented in this paper are motivated by the ideas of the sphere-constrained search strategy of the Fincke-Pohst algorithm [3]. There have been several modifications of the original sphere-constrained search strategy that may suggest further research directions. For instance, it could be beneficial to explore the possibility of applying the idea of statistical tree pruning of [15] to the JDD-ML and JDD-MAP algorithms. Essentially, one might decide to accept suboptimal solutions of the joint detection and decoding problems in exchange for decreasing the computational complexity. Such results would extend practical feasibility of the algorithms presented in this paper to a wider class of block codes and system parameters.

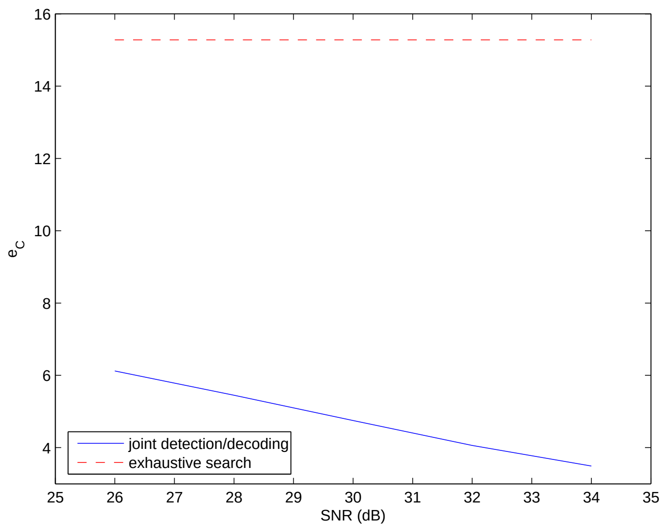

Figure 11: *The expected complexity exponent of the joint detection and decoding algorithm for the Reed-Solomon* $(15, 11)$ *code, and the complexity exponent of exhaustive search.*

A Appendix: A greedy algorithm for transforming generator matrix

Transformation of the $m \times k$ matrix $\mathbf{G}^T$ to the block upper-triangular form of Figure 3 is performed according to the following procedure:

1. Set $i = 1$, $G_i^T = \mathbf{G}^T$, $m_i = m$, $k_i = k$.
2. Search for and group together identical rows of matrix G_i^T; assume that the largest such group has $d \geq 1$ rows.
3. Permute the rows from the largest group found in step 2 to the bottom of G_i^T. (If there is more than one group with d identical rows, arbitrarily choose the group that will be permuted).
4. Using additions of rows, transform the bottom d rows in G_i^T so that they have maximum possible number of leading zeros. Denote the number of such leading zeros by j.
5. Increase $i = i + 1$; set $m_i = m - d$, $k_i = j$.
6. If both $m_i > 1$ and $k_i > 1$, denote the $m_i \times k_i$ left-upper submatrix of G_{i-1}^T by G_i^T and go to 2. Otherwise, use G_i^T, $i = 1, 2, \ldots$, to reassemble G^T.

Clearly, the operations that are allowed in the process of transforming $\mathbf{G}^T$ to the block upper-triangular form of Figure 3 are permutations and additions of rows. Therefore, the resulting matrix G generates the

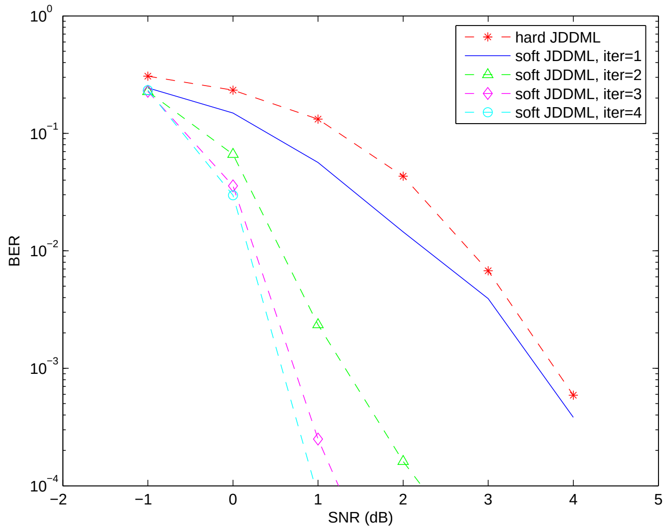

Figure 12: *Performance of the iterative decoding scheme employing the JDD-MAP algorithm. The system has outer convolutional and inner Golay code.*

same code as the starting $\mathbf{G}$, even though a particular information vector $\mathbf{b}$ may, in general, result in a different codeword upon encoding.

On another note, there is no guarantee that this construction yields G^T which is the best computationally, i.e., G^T for which the JDD-ML algorithm has the smallest complexity. Hence, we refer to the above algorithm as being greedy.

References

[1] E. Agrell, A. Vardy, and K. Zeger, "Closest point search in lattices," *IEEE Trans. on Info. Theory*, 2002.

[2] E. Viterbo and J. Boutros, "A universal lattice decoder for fading channels," *IEEE Trans. on Info. Theory*, vol. 45, pp. 1639–1642, July 2000.

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

[4] M. O. Damen, A. Chkeif, and J.-C. Belfiore, "Lattice code decoder for space-time codes," *IEEE Comm. Let.*, pp. 161–163, May 2000.

[5] B. Hassibi and H. Vikalo, “On sphere decoding algorithm. I. Expected complexity,” *IEEE Trans. on Sig. Proc.*, vol. 53, no. 8, August 2005, pp. 2806 - 2818.

[6] H. Vikalo and B. Hassibi, “On sphere decoding algorithm. II. Generalizations, second-order moments, and applications to communications,” *IEEE Trans. on Sig. Proc.*, vol. 53, no. 8, August 2005, pp. 2819 - 2834.

[7] H. Vikalo and B. Hassibi, “On joint ML detection and decoding for linear block codes,” in *Proc. IEEE Intern. Symposium on Info. Theory*, pp. 275 - 275, Yokohama, Japan, 2003.

[8] B. Hassibi and B. M. Hochwald, “High-rate codes that are linear in space and time,” in *IEEE Trans. on Info. Theory*, vol. 48, no. 7, pp. 1804-1824, July 2002.

[9] B. Hochwald and S. ten Brink, “Achieving near-capacity on a multiple-antenna channel,” *IEEE Trans. on Comm.*, 2003.

[10] H. Vikalo, B. Hassibi, and T. Kailath, “Iterative decoding for mimo channels via modified sphere decoder,” *IEEE Trans. on Wireless Comm.*, November 2004.

[11] R. J. McEliece, *The Theory of Information and Coding*, Cambridge University Press, 2nd ed., 2002.

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

[13] R. Koetter and A. Vardy, “Algebraic soft-decision decoding of Reed-Solomon codes,” *IEEE Trans. on Info. Theory*, vol. 49, pp. 2809-2825, November 2003.

[14] M. Fossorier and S. Lin, “Soft-Decision Decoding of Linear Block Codes Based on Ordered Statistics,” *IEEE Trans. on Info. Theory*, vol. 41, pp.1379-1396, September 1995.

[15] R. Gowaikar and B. Hassibi, “Efficient maximum-likelihood decoding via statistical pruning,” *submitted to IEEE Trans. on Sig. Proc.*, 2005.