12-1991

# Efficient Maximum-Likelihood Soft-Decision Decoding of Linear Block Codes Using Algorithm A

Yunghsiang S. Han
*Syracuse University*

Carlos R.P. Hartmann
*Syracuse University*, chartman@syr.edu

Chih-Chieh Chen

SU-CIS-91-42

# *Efficient Maximum-Likelihood Soft-Decision Decoding of Linear Block Codes Using Algorithm A**

Yunghsiang S. Han, Carlos R. P. Hartmann
and Chih-Chieh Chen

December 1991

*School of Computer and Information Science*
*Syracuse University*
*Suite 4-116, Center for Science and Technology*
*Syracuse, New York 13244-4100*

# Efficient Maximum-Likelihood Soft-Decision Decoding of Linear Block Codes Using Algorithm A*

Yunghsiang S. Han[1]

Carlos R. P. Hartmann[2]

Chih-Chieh Chen[3]

[1]Y. S. Han is with the School of Computer and Information Science at Syracuse University, Syracuse, NY 13244-4100 (e-mail: yshan@top.cis.syr.edu).

[2]C. R. P. Hartmann is with the School of Computer and Information Science at Syracuse University, Syracuse, NY 13244-4100 (e-mail: hartmann@top.cis.syr.edu).

[3]C.-C. Chen is with the Department of Electrical and Computer Engineering, Johns Hopkins University, Baltimore, MD 21218 (e-mail: johnny@jhunix.hcf.jhu.edu). Mr. Chen participated in this research during the summer of 1991 while he was working under the Syracuse Center of Computational Science Research Experience for Undergraduate Program grant from the National Science Foundation.

This work used the computational facilities of the Northeast Parallel Architectures Center (NPAC) at Syracuse University.

## Abstract

In this report we present a novel and efficient maximum-likelihood soft-decision decoding algorithm for linear block codes. The approach used here is to convert the decoding problem into a search problem through a graph which is a trellis for an equivalent code of the transmitted code. Algorithm $A^*$, which uses a priority-first search strategy, is employed to search through this graph. This search is guided by an evaluation function $f$ defined to take advantage of the information provided by the received vector and the inherent properties of the transmitted code. This function $f$ is used to drastically reduce the search space and to make the decoding efforts of this decoding algorithm adaptable to the noise level. Simulation results for the (48, 24) and the (72, 36) binary extended quadratic residue codes and the (128, 64) binary extended BCH code are given to substantiate the above claim.

# Contents

# 1 Introduction

Block codes and convolutional codes are two well-known error-control techniques for reliable transmission of digital information over noisy communication channels. Long linear block codes with coding gains far superior to those of convolutional codes have been known for many years. However, these block codes have not been used in practice for lack of an efficient soft-decision decoding algorithm. Soft-decision decoding can provide about 2 $dB$ of additional coding gain when compared to hard-decision decoding.

This report deals with the *maximum-likelihood soft-decision decoding of linear block codes.* By *maximum-likelihood decoding*, we mean a decoding algorithm which minimizes the probability of decoding to an incorrect codeword when all codewords have equal probability of being transmitted. By *soft-decision* we mean that the decoding algorithm can use real numbers (e.g., the analog output of filters matched to the signals) associated with every component of the codeword in the decoding procedure.

Our approach to the maximum-likelihood soft-decision decoding of linear block codes is to convert this problem into a search problem through a graph which is a trellis for a code equivalent to the transmitted code. To search through this graph we use Algorithm $A^*$ which uses a priority-first search strategy. It is widely used in Artificial Intelligence search problems [17]. The use of this algorithm for decoding drastically reduces the search space and results in an efficient optimal soft-decision decoding algorithm for linear block codes. Furthermore, the decoding efforts of this decoding algorithm are adaptable to the noise level.

In Section 2 we review maximum-likelihood decoding of linear block codes and describe Algorithm $A^*$. In the next section we present our decoding algorithm and give the simulation results for the (48,24) and (72,36) binary extended quadratic residue codes, and (128,64) binary extended BCH code. Concluding remarks and an extension of our decoding algorithm to convolutional codes are presented in Section 4.

## 2 Preliminaries

Let $\boldsymbol{V}_{n,q}$ be the set of all $n$-tuples over $\boldsymbol{GF(q)}$. A $q$-ary $(n,k)$ linear block code $\boldsymbol{C}$ is a subspace of $\boldsymbol{V}_{n,q}$ of dimension $k$. $\boldsymbol{C}$ can be characterized by a generator matrix $\boldsymbol{G}$ or by a parity-check matrix $\boldsymbol{H}$. Any set of $k$ linearly independent vectors in $\boldsymbol{C}$ can be used as the rows of $\boldsymbol{G}$. On the other hand, any set of $n-k$ linearly independent vectors in $\boldsymbol{C}^{\perp}$ (null space of $\boldsymbol{C}$) can be used as the rows of $\boldsymbol{H}$. Thus, a vector in $\boldsymbol{V}_{n,q}$ is a codeword in $\boldsymbol{C}$ if and only if it is a linear combination over $\boldsymbol{GF(q)}$ of the rows of $\boldsymbol{G}$. Therefore, a codeword in $\boldsymbol{C}$ can be written as $\boldsymbol{c} = \boldsymbol{u} \cdot \boldsymbol{G}$ where $\boldsymbol{u}$ is a $k$-tuple over $\boldsymbol{GF(q)}$. Since $\boldsymbol{C}$ is the null space of $\boldsymbol{C}^{\perp}$, any vector $\boldsymbol{v} \in \boldsymbol{V}_{n,q}$ is a codeword of $\boldsymbol{C}$ iff it is orthogonal to every row of $\boldsymbol{H}$, that is, $\boldsymbol{v} \cdot \boldsymbol{H}^T = \boldsymbol{0}$.

Let $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$, $c_j \in \boldsymbol{GF(q)}$ be a codeword of $\boldsymbol{C}$ transmitted over a time-discrete memoryless channel with output alphabet $\boldsymbol{B}$. Furthermore, let $\boldsymbol{r} = (r_0, r_1, \ldots, r_{n-1})$, $r_j \in \boldsymbol{B}$ denote the received vector, and let $\hat{\boldsymbol{c}}$ be an estimate of the transmitted codeword $\boldsymbol{c}$.

The *maximum-likelihood decoding rule* (**MLD** rule) for a time-discrete memoryless channel can be formulated as:

$$\text{set } \hat{\boldsymbol{c}} = \boldsymbol{c}_{\ell} \text{ where } \boldsymbol{c}_{\ell} = (c_{\ell 0}, c_{\ell 1}, \ldots, c_{\ell(n-1)}) \in \boldsymbol{C} \text{ and}$$

$$\prod_{j=0}^{n-1} \boldsymbol{Pr}(r_j | c_{\ell j}) \geq \prod_{j=0}^{n-1} \boldsymbol{Pr}(r_j | c_{ij}) \text{ for all } \boldsymbol{c}_i = (c_{i0}, c_{i1}, \ldots, c_{i(n-1)}) \in \boldsymbol{C}.$$

We now give the **MLD** rule for a binary linear code $\boldsymbol{C}$ transmitted over the Additive White Gaussian Noise (**AWGN**) channel using antipodal signaling. In this case, the $j^{th}$ component of a codeword $\boldsymbol{c} = (c_0, c_1, \ldots, c_j, \ldots, c_{n-1})$ of $\boldsymbol{C}$ is transmitted as

$$(-1)^{c_j}\sqrt{E},$$

where $E$ is the signal energy per channel bit. So the $j^{th}$ component of the received vector $\boldsymbol{r} = (r_0, r_1, \ldots, r_j, \ldots, r_{n-1})$ is

$$r_j = (-1)^{c_j}\sqrt{E} + e_j,$$

where $e_j$ is the noise sample of a Gaussian process with single-sided noise power per hertz

$N_0$. The variance of $e_j$ is $N_0/2$. In this case the **MLD** rule can be written as

$$\text{set } \hat{\boldsymbol{c}} = \boldsymbol{c}_\ell, \text{ where } \boldsymbol{c}_\ell \in \boldsymbol{C} \text{ and}$$

$$\sum_{j=0}^{n-1} \left(r_j - (-1)^{c_{\ell j}}\sqrt{E}\right)^2 \leq \sum_{j=0}^{n-1} \left(r_j - (-1)^{c_{ij}}\sqrt{E}\right)^2 \quad \text{for all } \boldsymbol{c}_i \in \boldsymbol{C}.$$

In the special case where the codewords of $\boldsymbol{C}$ have equal probability of being transmitted, the **MLD** rule minimizes the error probability.

One way to implement the **MLD** rule is to calculate $\boldsymbol{Pr}(\boldsymbol{r}|\boldsymbol{c}_i) = \prod_{j=0}^{n-1} \boldsymbol{Pr}(r_j|c_{ij})$ for every codeword in $\boldsymbol{C}$ and select the codeword that maximizes it. In practice this can be done only for those codes with a small number of codewords, that is, low rate codes or middle-to-high rate codes with short block length.

In 1979 Hwang [12] showed that it is possible to select a codeword maximizing $\boldsymbol{Pr}(\boldsymbol{r}|\boldsymbol{c}_i)$ without calculating it for every codeword in a binary code $\boldsymbol{C}$. He proved that if a codeword $\boldsymbol{c}_{i1} = \boldsymbol{c}_{i2} \oplus \boldsymbol{c}_{i3}$ where $\boldsymbol{c}_{i2}$ and $\boldsymbol{c}_{i3}$ are disjoint codewords, then we can select a codeword maximizing $\boldsymbol{Pr}(\boldsymbol{r}|\boldsymbol{c}_i)$ without directly calculating $\boldsymbol{Pr}(\boldsymbol{r}|\boldsymbol{c}_{i1})$. Recently it has been shown that if the **MLD** rule is implemented using Hwang's technique, then the codewords that can be dropped from the computation are only those satisfying the above property [14, 16].

In 1980 Hwang [13] proposed another approach to reduce the number of codewords that need to be considered when applying the MLD rule. However, since the $k$ most "reliable" positions of the received vector may not be linearly independent, it is simple to design an example where the procedure proposed in [13] will fail to start. Such an example is given in Appendix A.

Several researchers [6, 21, 18] have presented a technique for decoding linear block codes that converts the decoding problem into a graph-search problem on a trellis derived from the parity-check matrix of the code. Thus the MLD rule can be implemented by applying the Viterbi Algorithm [20] to this trellis which has at most $\min\left(q^k, q^{n-k}\right)$ states. Therefore, in practice this breadth-first search scheme can be applied only to codes with small redundancy, that is, small $n-k$ or codes with a small number of codewords.

When the decoding problem is converted into a graph-search problem, we are interested in finding a path from the start node representing the initial condition to a goal node that

represents the termination condition. This path will optimize some criterion that leads us to construct a codeword that maximizes $\boldsymbol{Pr}(\boldsymbol{r}|\boldsymbol{c}_i)$, where $\boldsymbol{c}_i \in \boldsymbol{C}$.

Thus, the decoding problem has been mapped to a more general graph-search problem. In this graph each arc is assigned a cost and the cost of a path is the sum of the costs of the arcs connecting the nodes in this path. The problem is how to find an optimal path from the start node to a goal node, that is, a path with minimal (maximal) cost. The algorithm $A^*$, widely used in Artificial Intelligence, is an efficient procedure for finding an optimal path if one exists in a graph.

In order to more easily describe Algorithm $A^*$, we first give a general graph-searching procedure as presented in [17]:

Procedure **GRAPHSEARCH**

1. Create a *search graph*, $\mathcal{G}$, consisting solely of the start node, $s$. Put $s$ on a list called *OPEN*.

2. Create a list called *CLOSED* that is initially empty.

3. LOOP: if *OPEN* is empty, exit with failure.

4. Select the first node on *OPEN*, remove it from *OPEN*, and put it on *CLOSED*. Call this node $m$.

5. If $m$ is a goal node, exit successfully with the solution obtained by tracing a path along the pointers from $m$ to $s$ in $\mathcal{G}$. (Pointers are established in Step 7.)

6. Expand node $m$, generating the set, $M$, of its successors that are not ancestors of $m$. Install these members of $M$ as successors of $m$ in $\mathcal{G}$.

7. Establish a pointer to $m$ from those members of $M$ that were not already in $\mathcal{G}$ (i.e., not already on either *OPEN* or *CLOSED*). Add these members of $M$ to *OPEN*. For each member of $M$ that was already on *OPEN* or *CLOSED*, decide whether or not to redirect its pointer to $m$. For each member of $M$ already on *CLOSED*, decide for each of its descendants in $\mathcal{G}$ whether or not to redirect its pointer.

8. Reorder the list *OPEN*, either according to some arbitrary scheme or according to heuristic merit.

9. Go LOOP.

If the graph being searched is not a tree, it is possible that some of the elements of set $M$ have already been visited—that is, they are already on list OPEN or list CLOSED. The problem of determining whether a newly generated node is on these lists can be computationally very expensive. For this reason we may decide to avoid making this test, in which case the search tree may contain several repeated nodes. These node repetitions lead to redundant successor computations and there is a trade-off between the computation cost for testing for repeated nodes and the computation cost for generating a larger search tree. In steps 6 and 7 of procedure **GRAPHSEARCH**, testing for repeated nodes is performed.

In an uninformed search procedure no heuristic information from the problem has been used in reordering the list OPEN in Step 8. In this case, the two well-known search methods are the breadth-first and depth-first. However, these methods are exhaustive in nature, and thus in practice are applicable only to graphs with small numbers of nodes and paths.

In many cases it is possible to use some inherent properties of a problem to help reduce the search. The search procedure using this information is called a *heuristic search method.* In many situations it is possible to specify heuristics that reduce considerably the search effort without compromising the optimality of the solution.

One of the well-known heuristic search methods that guarantee to find the optimal solution if one exists is the Algorithm $A^*$ [17]. The description of $A^*$ given here is taken from [17]. $A^*$ uses a cost function called *evaluation function* $f$ to guide the search through the graph. This function $f$ is computed for every node that is added to list OPEN in Step 7 of the procedure **GRAPHSEARCH**. In Step 8 of this procedure, we reorder the list OPEN according to the value of the function $f$. From now on, in order to simplify the description of $A^*$, we assume that an optimal path is one that minimizes the cost function.

For every node $m$, we define the evaluation function $f$ so that its value $f(m)$ at node $m$ estimates the cost of the minimum cost path that goes through node $m$. $f(m)$ is computed

as

$$f(m) = g(m) + h(m),$$

where $g(m)$ estimates the cost of the minimal cost path from the start node $s$ to node $m$, and $h(m)$ estimates the cost of the minimal cost path from node $m$ to a goal node.

In $A^*$, the next node to be expanded is the one with the smallest value of $f$ on the list OPEN since this node imposes the least severe constraints.

Similarly, let $f^*$ be a function such that $f^*(m)$ at any node $m$ is the actual cost of a minimum cost path that goes through node $m$. Analogously,

$$f^*(m) = g^*(m) + h^*(m),$$

where $g^*(m)$ is the actual cost of a minimum cost path from the start node $s$ to node $m$, and $h^*(m)$ is the actual cost of a minimum cost path from node $m$ to a goal node.

$A^*$ requires that $g(m) \geq g^*(m)$ and $h(m) \leq h^*(m)$ for every node $m$ of the graph. These requirements guarantee that $A^*$ will find a minimum cost path if one exists; however, if the graph is finite, then the only condition that must be satisfied to guarantee optimality is $h(m) \leq h^*(m)$ for every node $m$ of the graph [17].

An obvious choice for $g(m)$ is the cost of the path in the search tree from the start node $s$ to node $m$ given by summing all the arc costs encountered while constructing the minimum cost path from the start node $s$ to node $m$. Note that this path is the lowest cost path from the start node $s$ to node $m$ found so far by the algorithm. The value of $g(m)$ may decrease if the search tree is altered in Step 7 of procedure **GRAPHSEARCH**. From now on we assume that function $g$ is calculated in this way. In this case $g(m) \geq g^*(m)$ for every node $m$ of the graph. Furthermore, if $h(m) = 0$ for any node $m$, then $A^*$ becomes a version of Dijkstra's algorithm [9].

To define $h(m) \leq h^*(m)$, we use the properties of the problem. It can be shown [17] that if we have two evaluation functions $f_1(m) = g_1(m) + h_1(m)$ and $f_2(m) = g_2(m) + h_2(m)$ satisfying $h_1(m) < h_2(m) \leq h^*(m)$ for every node $m$, the $A^*$ using evaluation function $f_2$ will never expand more nodes than the $A^*$ using evaluation function $f_1$. Furthermore, if there exists a unique optimal path, then the above results hold when $h_1(m) \leq h_2(m) \leq h^*(m)$ is

satisfied for every node $m$. Also, if $h(m) > 0$ for any node $m$, then $A^*$, using this function $h$, will never expand more nodes than the above version of Dijkstra's algorithm.

The monotone restriction is a reasonable restriction that when imposed on $h$ can substantially decrease the computation time and storage of $A^*$. In [17], the function $h$ is said to satisfy the monotone restriction if and only if for all nodes $m_i$ and $m_j$, such that node $m_j$ is a successor of node $m_i$,

$$0 \leq h(m_i) - h(m_j) \leq c(m_i, m_j)$$

with $h(t) = 0$, where $t$ is any goal node and $c(m_i, m_j)$ is the arc cost between node $m_i$ and node $m_j$.

If the monotone restriction is satisfied, then it can be shown [17] that $A^*$ has already found an optimal path from the start node to the node it selects to expand. Thus there is no need for $A^*$ to check if the newly generated nodes are on the list CLOSED and we do not have to store this list. Furthermore, we do not have to update the parentage in the search tree of any successors of the node $A^*$ selects to expand. Also, if the monotone restriction is satisfied, the $f$ values of the sequence of nodes expanded by $A^*$ is nondecreasing [17].

In the proof of the above results [17], the conditions that are imposed by the monotone restriction, namely $0 \leq h(m_i) - h(m_j)$ and $h(t) = 0$, have not been used. So the only requirement for this proof is that

$$h(m_i) \leq h(m_j) + c(m_i, m_j). \tag{1}$$

We will show that the $h$ function we use in the next section will satisfy this inequality, so we can still use the above result to speed up the decoding procedure. It is easy to find an example to show that this $h$ function does not satisfy $0 \leq h(m_i) - h(m_j)$.

Another property of $A^*$ [17, Prob. 2.6] that will be used in our decoding algorithm is as follows: Algorithm $A^*$ still finds the optimal path (if one exists) if it removes from list OPEN any node $m$ for which $f(m) > UB$, where $UB$ is an upper bound on the cost of an optimal path.

From the description of $A^*$ it is clear that the most important factor in the efficiency of $A^*$ is the selection of the heuristic function $h$ and, consequently, the evaluation function $f$.

# 3 Decoding Algorithm

In order to simplify the description of our algorithm we restrict ourselves to binary linear codes transmitted over the **AWGN** channel using antipodal signaling. Thus, as stated earlier, the $j^{th}$ component of $\boldsymbol{c} = (c_0, c_1, \ldots, c_j, \ldots, c_{n-1}) \in \boldsymbol{C}$ is transmitted as $(-1)^{c_j}\sqrt{E}$ and the $j^{th}$ component of the received vector $\boldsymbol{r} = (r_0, r_1, \ldots, r_j, \ldots, r_{n-1})$ is $r_j = (-1)^{c_j}\sqrt{E} + e_j$.

Our decoding algorithm, guided by an evaluation function $f$, searches through a graph that is a trellis for a code $\boldsymbol{C}^*$, which is equivalent to code $\boldsymbol{C}$. $\boldsymbol{C}^*$ is obtained from $\boldsymbol{C}$ by permuting the positions of codewords of $\boldsymbol{C}$ in such a way that the first $k$ positions of codewords in $\boldsymbol{C}^*$ correspond to the "most reliable linearly independent" positions in the received vector $\boldsymbol{r}$. In Appendix B we give an algorithm to obtain $\boldsymbol{G}^*$ from $\boldsymbol{G}$. $\boldsymbol{G}^*$ is a generator matrix of $\boldsymbol{C}^*$ whose first $k$ column forms a $k \times k$ identity matrix. The time complexity of this algorithm is also discussed in this appendix.

In our decoding algorithm the vector $\boldsymbol{r}^* = (r_0^*, r_1^*, \ldots, r_{n-1}^*)$ is used as the "received vector." It is obtained by permuting the positions of $\boldsymbol{r}$ in the same manner in which the columns of $\boldsymbol{G}$ can be permuted to obtain $\boldsymbol{G}^*$.

## 3.1 Construction of trellis

We now give a short description of a trellis [1] for the code $\boldsymbol{C}^*$ where the search will be performed. We remark here that even though we will describe the complete trellis, our decoding algorithm will construct only a very small subgraph of this trellis during the decoding procedure. Let $\boldsymbol{H}^*$ be a parity-check matrix of $\boldsymbol{C}^*$, and let $\boldsymbol{h}_i{}^*$, $0 \leq i < n$ be the column vectors of $\boldsymbol{H}^*$. Furthermore, let $\boldsymbol{c}^* = (c_0^*, c_1^*, \ldots, c_{n-1}^*)$ be a codeword of $\boldsymbol{C}^*$. With respect to this codeword, we recursively define the states $\boldsymbol{s}_t$, $-1 \leq t < n$, as follows:

$$\boldsymbol{s}_{-1} = \boldsymbol{0}$$

and

$$\boldsymbol{s}_t = \boldsymbol{s}_{t-1} + c_t^* \boldsymbol{h}_t^* = \sum_{i=0}^{t} c_i^* \boldsymbol{h}_i^*, \quad 0 \leq t < n.$$

Clearly, $\boldsymbol{s}_{n-1} = \mathbf{0}$ for all codewords of $C^*$. The above recursive equation can be used to draw a trellis diagram with at most $\min(2^k, 2^{n-k})$ states. In this trellis, $\boldsymbol{s}_{-1} = \mathbf{0}$ identifies the start node which is at level $-1$; $\boldsymbol{s}_{n-1} = \mathbf{0}$ identifies the goal node which is at level $n-1$; and each state $\boldsymbol{s}_t, 0 \leq t < n-1$ identifies a node at level $t$. Furthermore, each transition (arc) is labelled with the appropriate codeword bit $c_t^*$. A more detailed description of a trellis for a linear block code can be found in [21]. Note that the trellis defined here corresponds to the expurgated trellis of [21].

## 3.2 Evaluation function

As we pointed out before, the selection of evaluation function $f$ is of the utmost importance, since it determines the search effort of $A^*$. We now describe the function $f$ we use in our decoding algorithm.

In order to define function $f$, we need first to specify the arc costs. In the trellis of $\boldsymbol{C}^*$, the cost of the arc from $\boldsymbol{s}_{t-1}$ to $\boldsymbol{s}_t = \boldsymbol{s}_{t-1} + c_t^* \boldsymbol{h}^*_t$ is assigned the value $(r_t^* - (-1)^{c_t^*}\sqrt{E})^2$. Thus the solution of the decoding problem is converted into finding a path from the start node to the goal node, that is, a codeword $\boldsymbol{c}^* = (c_0^*, c_1^*, \ldots, c_{n-1}^*)$ such that $\sum_{i=0}^{n-1}(r_i^* - (-1)^{c_i^*}\sqrt{E})^2$ is minimum among all paths from the start node to the goal node.

Now we define function $f$ for every node $m$ in the trellis as follows:

$$f(m) = g(m) + h(m).$$

As previously noted, $g(m)$ is the lowest cost path from the start node to node $m$ found so far by the algorithm, where the cost of a path from the start node to node $m$ is obtained by summing all the arc costs encountered while constructing this path.

We now define a function $h$ which must satisfy $h(m) \leq h^*(m)$ for every node $m$. Recall that $h^*(m)$ is the cost of a minimum cost path from node $m$ to the goal node.

1. For nodes at level $\ell, k-1 \leq \ell < n$:

   Because of the linear property of $\boldsymbol{C}^*$ and the fact that the first $k$ columns of $\boldsymbol{G}^*$ are linearly independent, there is only one path $\boldsymbol{P}_{m'}$ from any node $m'$ at level $k-1$ to the

goal node. Furthermore, we can easily determine the labels $c_k^*, c_{k+1}^*, \ldots, c_{n-1}^*$ of this path using $\boldsymbol{G}^*$ and calculate its cost $\sum_{i=k}^{n-1} \left(r_i^* - (-1)^{c_i^*}\sqrt{E}\right)^2$. In view of the above fact, we define function $h$ as follows:

$$h(m) = \sum_{i=\ell+1}^{n-1} \left(r_i^* - (-1)^{c_i^*}\sqrt{E}\right)^2,$$

where $c_{\ell+1}^*, c_{\ell+2}^*, \ldots, c_{n-1}^*$ are the labels of the only path $\boldsymbol{P}_m$ from node $m$ to the goal node.

Note that if node $m$ is the goal node, then $h(m) = 0$. Furthermore, $h(m) = h^*(m)$ since there is only one path from node $m$ to the goal node and $h(m)$ is the cost of this path. The time complexity for calculating $h(m)$ for node $m$ at level $\ell, k-1 \leq \ell < n$ is discussed in Appendix F.

2. For nodes at level $\ell, -1 \leq \ell < k-1$:

   In order to define a function $h$ which is a "good" estimator of $h^*$ we must use properties of the linear block code which are invariant under any permutation of the positions of the codewords. Here our function $h$ is defined to take into consideration the fact that codewords of $\boldsymbol{C}$ can only have some specific Hamming weights.

   Let $w_0 < w_1 < \cdots < w_I$ be the $I+1$ distinct Hamming weights that codewords of $\boldsymbol{C}$ may have. Thus these are the only Hamming weights that codewords of $\boldsymbol{C}^*$ may have.

   Let $c_0^*, c_1^*, \ldots, c_\ell^*$ be the labels of the lowest cost path $\boldsymbol{P}'_m$ from the start node to node $m$ which is at level $\ell$ found so far by the algorithm, and let $W(\boldsymbol{P}'_m)$ be the number of labels of $\boldsymbol{P}'_m$ whose values are 1.

   Furthermore, let $\boldsymbol{v} = (v_{\ell+1}, v_{\ell+2}, \ldots, v_{n-1})$ be a binary $(n-\ell-1)$-tuple. Denote by $W_H(\boldsymbol{v})$ the Hamming weight of $\boldsymbol{v}$. Now, define the set

$$T(m) = \left\{\boldsymbol{v} | W_H(\boldsymbol{v}) \in \left\{w_i - W(\boldsymbol{P}'_m) | 0 \leq i \leq I\right\}\right\}.$$

   Note that $0 \leq W_H(\boldsymbol{v}) \leq (n-\ell-1)$ for all $\boldsymbol{v} \in T(m)$. We will show that $T(m) \neq \emptyset$ in

Appendix C. Finally, we define $h$ as

$$h(m) = \min_{\boldsymbol{v} \in T(m)} \left\{ \sum_{i=\ell+1}^{n-1} \left( r_i^* - (-1)^{v_i} \sqrt{E} \right)^2 \right\}.$$

Obviously, $h(m) \leq h^*(m)$ for any node $m$ in the trellis.

It is very important that the time complexity for calculating $h(m)$ be "reasonable," for otherwise the time taken by the decoding algorithm is spent calculating $h(m)$, even though there are only a few nodes to be visited (open) in the trellis. In Appendix C we present an algorithm to calculate $h(m)$ for node $m$ at level $\ell$, $-1 \leq \ell < k-1$ whose time complexity is $O(n)$.

For long block codes it may be impossible to determine the set, HW, of different Hamming weights the codewords may have. However, our algorithm will still find the optimal solution even if in the computation of function $h$ the algorithm considers all the Hamming weights of any superset of HW. The algorithm using a superset of HW may visit more nodes than that using HW. Furthermore, in most cases the received vector is closed to a unique codeword. In this case, as pointed out in Section 2, the algorithm will not open fewer nodes if it uses a proper superset of HW instead of HW in the computation of function $h$.

## 3.3 Speed-up techniques

In this subsection we present some properties of the decoding algorithm that can be used to speed up the decoding procedure.

In Appendix D we show that our $h$ function satisfies the following property:

$$h(m_i) \leq h(m_j) + c(m_i, m_j),$$

where node $m_j$ is a successor of node $m_i$ and $c(m_i, m_j)$ is our arc cost from node $m_i$ to node $m_j$. Then, as we pointed out before, we do not need to store the list CLOSED and we do not have to update the parentage in the search tree of any successors of the node that our algorithm selects to expand.

Another property that can be used to speed up the decoding process (to be shown in Appendix E) is the fact that when our algorithm expands a node $m$ at level $\ell < k-2$, we need to compute the value of function $f$ for only one of its successors. This is because the value of function $f$ for the other successor is equal to that of node $m$ and we can easily determine which successor has the value $f(m)$. Thus our algorithm is a depth-first search type Algorithm $A^*$.

Furthermore, since our function $h$ satisfies Inequality 1, by the remark in the previous section, the $f$ values of the sequence of nodes expanded by our algorithm is nondecreasing. Let node $m_1$ at level $\ell < k-2$ be the first node of list OPEN. Consider the sequence of nodes that the algorithm will follow from node $m_1$ to node $m_2$ which is at level $k-2$. Due to the above properties, the value of the function $f$ at every one of these nodes is equal to $f(m_1)$. Furthermore, the labels of the path corresponding to this sequence of nodes is determined when the algorithm calculates $h(m_1)$ (see Appendix E). Thus, we do not have to visit the nodes of this sequence. This reduces considerably the total number of nodes visited.

Our algorithm will search the trellis only up to level $k-1$ since we can construct the only path from any node $m$ at level $k-1$ to the goal node using $\boldsymbol{G}^*$. The labels of the combined paths from the start node to node $m$, and from node $m$ to the goal node, correspond to a codeword. So the cost of this path, which is equal to $f(m)$, can be used as an upper bound on the cost of an optimal path. As noted in Section 2, we can use this upper bound to reduce the size of list OPEN. Furthermore, since in an expurgated trellis every path from the start node reaches the goal node, and the $f$ values of the sequence of nodes expanded by our algorithm is nondecreasing, then we only need to keep one node on list OPEN whose $f$ value is equal to the upper bound.

Now we discuss another property of linear block codes that allows us to reduce the search procedure for our algorithm. Let $\boldsymbol{R}^n$ be the set of all $n$-tuples over $\boldsymbol{R}$, the set of real numbers, and let $\boldsymbol{c}^* = (c_0^*, c_1^*, \ldots, c_{n-1}^*)$ be a codeword of $\boldsymbol{C}^*$. Denote by $\overrightarrow{\mathbf{0}\boldsymbol{v}_{c^*}}$ the geometric vector from the point $\mathbf{0} = (0, 0, \ldots, 0) \in \boldsymbol{R}^n$ to the point $\boldsymbol{v}_{c^*} = ((-1)^{c_0^*}\sqrt{E}, (-1)^{c_1^*}\sqrt{E}, \ldots, (-1)^{c_{n-1}^*}\sqrt{E}) \in \boldsymbol{R}^n$. Analogously, denote by $\overrightarrow{\mathbf{0}\boldsymbol{r}^*}$ the geometric vector from the point $\mathbf{0}$ to the point $\boldsymbol{r}^* = (r_0^*, r_1^*, \ldots, r_{n-1}^*) \in \boldsymbol{R}^n$. Furthermore, let ANG( $\overrightarrow{\mathbf{0}\boldsymbol{v}_{c^*}}$, $\overrightarrow{\mathbf{0}\boldsymbol{r}^*}$) be the smallest angle between

$\overrightarrow{\mathbf{0}\boldsymbol{v}_{c^*}}$ and $\overrightarrow{\mathbf{0}\boldsymbol{r}^*}$. Let $\alpha = \sin^{-1}\sqrt{\frac{d_{\min}}{n}}$, where $d_{\min}$ is the minimum Hamming distance of $C^*$, which is the same as that of $C$. It is very simple to show that if $\mathrm{ANG}(\overrightarrow{\mathbf{0}\boldsymbol{v}_{c^*}}, \overrightarrow{\mathbf{0}\boldsymbol{r}^*}) \leq \alpha$, then $\sum_{i=0}^{n-1} \left(r_i^* - (-1)^{c_i^*}\sqrt{E}\right)^2$ is minimum over all codewords of $C^*$. Furthermore, it is easy to calculate $\mathrm{ANG}(\overrightarrow{\mathbf{0}\boldsymbol{v}_{c^*}}, \overrightarrow{\mathbf{0}\boldsymbol{r}^*})$. First, we multiply $\boldsymbol{r}^*$ by a scalar $\beta$ such that $\beta\boldsymbol{r}^*$ lies on the surface of the sphere of radius $\sqrt{nE}$ centered at $\mathbf{0}$. Thus

$$\mathrm{ANG}(\overrightarrow{\mathbf{0}\boldsymbol{v}_{c^*}}, \overrightarrow{\mathbf{0}\boldsymbol{r}^*}) = 2 \cdot \sin^{-1}\left(\frac{1}{2} \cdot \sqrt{\frac{\sum_{i=0}^{n-1}\left(\beta r_i^* - (-1)^{c_i^*}\sqrt{E}\right)^2}{nE}}\right).$$

Every time our algorithm constructs a codeword $\boldsymbol{c}^*$ we can check if $\mathrm{ANG}(\overrightarrow{\mathbf{0}\boldsymbol{v}_{c^*}}, \overrightarrow{\mathbf{0}\boldsymbol{r}^*}) \leq \alpha$ holds. If so, $\boldsymbol{c}^*$ is an optimal solution.

## 3.4 Simulation results

In the implementation of our decoding algorithm we have decided not to check for repeated nodes. In this situation the graph becomes a decision tree. Thus, we do not have to keep list CLOSED. Furthermore, list OPEN is always kept ordered according to the values $f$ of its nodes. An outline of our decoding algorithm under this condition is given in Appendix F. In this appendix we also apply the algorithm to a numerical example.

As derived in Appendix F, the time complexity and the space complexity of our algorithm are $O(n \times N(\boldsymbol{r}))$ and $O(n \times M(\boldsymbol{r}))$, respectively. Recall that

$N(\boldsymbol{r})$ = the number of nodes visited during the decoding of $\boldsymbol{r}$;

$M(\boldsymbol{r})$ = maximum number of nodes stored on list OPEN during the decoding of $\boldsymbol{r}$.

The values of $N(\boldsymbol{r})$ and $M(\boldsymbol{r})$ will strongly depend upon the signal to noise ratio (SNR). Up to now we do not have a "good" estimator of these values; however, they are upperbounded by $2^{k+1} - 1$. So, in the worst case, the time and space complexities of our algorithm are $O(n \times 2^k)$, which are, under the condition $k \leq (n-k)$, equal to those of Wolf's algorithm [21], which are $O(n \times \min(2^k, 2^{n-k}))$ [8].

In order to verify the performance of our algorithm we give simulation results for the (48,24) and the (72,36) binary extended quadratic residue codes, and the (128,64) binary extended BCH code. For the (48,24) and the (72,36) codes, we know HW. However, for the (128,64) code, we only know a superset of HW, namely, $\{x|$ ($x$ is even and $22 \leq x \leq 106$) or ($x = 0$) or ($x = 128$)$\}$.

First, we give simulation results for the (48,24) and the (72,36) codes. Both of theses codes have $d_{\min}$ equal to 12. These codes are obtained by adding an overall parity check to the (47,24) and the (71,36) binary quadratic residue codes, respectively. Quadratic residue codes are known to be very good codes that are very difficult to decode even when only hard-decision decoding is employed [4, 7, 5]. Some quadratic residue codes have been decoded by using information-set decoding algorithms [3]. However, these algorithms are sub-optimal, that is, do not implement the **MLD** rule. Thus, the only two non-exhaustive maximum-likelihood soft-decision decoding algorithms known to us that can be used to decode the (48,24) and the (72,36) codes are Wolf's algorithm [21] and Hwang's algorithm [12].

It is very hard for us to compare the performance of our algorithm with that of Hwang because he found the subset of codewords that must be stored for implementing the **MLD** rule only for very short codes [12, Table I]. However, we observe that the complexities of Wolf's algorithm are approximately the same as those of Hwang's for the codes presented in Table I of [12]. Thus, we will compare the performance of our algorithm to that of Wolf.

As pointed out before, we assume that antipodal signaling is used in the transmission so that the $j^{th}$ components of the transmitted codeword $\boldsymbol{c}$ and received vector $\boldsymbol{r}$ are

$$c_j = (-1)^{c_j}\sqrt{E} \quad \text{and} \quad r_j = (-1)^{c_j}\sqrt{E} + e_j,$$

respectively, where $E$ is the signal energy per channel bit and $e_j$ is a noise sample of a Gaussian process with single-sided noise power per hertz $N_0$. The variance of $e_j$ is $N_0/2$ and the SNR for the channel is $\gamma = E/N_0$. In order to account for the redundancy in codes of different rates, we used the SNR per transmitted information bit $\gamma_b = E_b/N_0 = \gamma n/k$ in our simulation.

The simulation results for the $(48, 24)$ binary extended quadratic residue code for $\gamma_b$ equal

to 2 $dB$, 3 $dB$, 4 $dB$, 5 $dB$, 6 $dB$, and 7 $dB$ are given in tables 1, 2, and 3. These results were obtained by simulating 17,000 samples for each SNR. Note that $2^{24} \approx 1.68 \times 10^7$.

Table 1: Simulation for the $(48, 24)$ code

| $\gamma_b$ | 2 $dB$ | | | 3 $dB$ | | | 4 $dB$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | max | min | ave | max | min | ave | max | min | ave |
| $N(\boldsymbol{r})$ | 19442 | 24 | 452 | 13997 | 24 | 165 | 9141 | 24 | 58 |
| $C(\boldsymbol{r})$ | 7338 | 1 | 134 | 5154 | 1 | 42 | 2996 | 1 | 11 |
| $M(\boldsymbol{r})$ | 2427 | 24 | 66 | 1491 | 24 | 35 | 1101 | 24 | 26 |

| $\gamma_b$ | 5 $dB$ | | | 6 $dB$ | | | 7 $dB$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | max | min | ave | max | min | ave | max | min | ave |
| $N(\boldsymbol{r})$ | 1377 | 24 | 30 | 551 | 24 | 25 | 183 | 24 | 25 |
| $C(\boldsymbol{r})$ | 356 | 1 | 3 | 164 | 1 | 2 | 34 | 1 | 2 |
| $M(\boldsymbol{r})$ | 159 | 24 | 25 | 66 | 24 | 25 | 47 | 24 | 25 |

$N(\boldsymbol{r})$ = the number of nodes visited during the decoding of $\boldsymbol{r}$;

$C(\boldsymbol{r})$ = number of codewords constructed in order to decide on the closest codeword to $\boldsymbol{r}$;

$M(\boldsymbol{r})$ = maximum number of nodes stored on list OPEN during the decoding of $\boldsymbol{r}$;

max = maximum value among 17,000 samples;

min = minimum value among 17,000 samples;

ave = average value among 17,000 samples.

Table 2: The number of samples for which ANG($\overrightarrow{\mathbf{0}\boldsymbol{v}_{c^*}}$, $\overrightarrow{\mathbf{0}\boldsymbol{r}^*}$) $\leq \alpha$ holds for the (48,24) code

| $\gamma_b$ | 2 $dB$ | 3 $dB$ | 4 $dB$ | 5 $dB$ | 6 $dB$ | 7 $dB$ |
|---|---|---|---|---|---|---|
| # | 423 | 1708 | 5128 | 10567 | 15224 | 16844 |

For high SNR the bit error probability can be estimated using the following formula [11]:

$$n_d\sqrt{d_{\min}/(4\pi nk\gamma_b)}\; e^{-(kd_{\min}\gamma_b/n)} \tag{2}$$

where $n_d$ is the number of codewords of Hamming weight $d_{\min}$.

Table 3: Bit error probability and coding gain for the (48,24) code

| $\gamma_b$ | 2 $dB$ | 3 $dB$ | 4 $dB$ | 5 $dB$ | 6 $dB$ | 7 $dB$ |
|---|---|---|---|---|---|---|
| $P_b$ | $9.60 \times 10^{-3}$ | $1.34 \times 10^{-3}$ | $8.95 \times 10^{-5}$* | $1.61 \times 10^{-6}$* | $1.06 \times 10^{-8}$* | $1.94 \times 10^{-11}$* |
| $CG$ | 2.36 | 3.55 | 4.45 | 5.35 | 5.95 | 6.40 |

$P_b$ = bit error probability;

$CG$ = coding gain ($dB$);

* Calculate using (2).

The simulation results for the $(72, 36)$ binary extended quadratic residue code for $\gamma_b$ equal to 2 $dB$, 3 $dB$, 4 $dB$, 5 $dB$, 6 $dB$, and 7 $dB$ are given in tables 4, 5, and 6. These results were obtained by simulating 17,000 samples for each SNR. Note that $2^{36} \approx 6.87 \times 10^{10}$.

Table 4: Simulation for the $(72, 36)$ code

| $\gamma_b$ | 2 $dB$ | | | 3 $dB$ | | | 4 $dB$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | max | min | ave | max | min | ave | max | min | ave |
| $N(\boldsymbol{r})$ | 1012728 | 37 | 11931 | 683567 | 36 | 2075 | 38326 | 36 | 326 |
| $C(\boldsymbol{r})$ | 383542 | 2 | 3750 | 241090 | 1 | 560 | 11716 | 1 | 64 |
| $M(\boldsymbol{r})$ | 130372 | 36 | 1413 | 88031 | 36 | 241 | 4288 | 36 | 56 |

| $\gamma_b$ | 5 $dB$ | | | 6 $dB$ | | | 7 $dB$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | max | min | ave | max | min | ave | max | min | ave |
| $N(\boldsymbol{r})$ | 24493 | 36 | 81 | 1758 | 36 | 41 | 368 | 36 | 37 |
| $C(\boldsymbol{r})$ | 6202 | 1 | 10 | 446 | 1 | 3 | 50 | 1 | 2 |
| $M(\boldsymbol{r})$ | 2514 | 36 | 38 | 366 | 36 | 37 | 41 | 36 | 37 |

Table 5: The number of samples for which $\mathrm{ANG}(\overrightarrow{\mathbf{0}\boldsymbol{v}_{c^*}}, \overrightarrow{\mathbf{0}\boldsymbol{r}^*}) \leq \alpha$ holds for the (72,36) code

| $\gamma_b$ | 2 $dB$ | 3 $dB$ | 4 $dB$ | 5 $dB$ | 6 $dB$ | 7 $dB$ |
|---|---|---|---|---|---|---|
| # | 0 | 4 | 35 | 377 | 2767 | 9323 |

Table 6: Bit error probability and coding gain for the (72,36) code

| $\gamma_b$ | 2 $dB$ | 3 $dB$ | 4 $dB$ | 5 $dB$ | 6 $dB$ | 7 $dB$ |
|---|---|---|---|---|---|---|
| $P_b$ | $7.13 \times 10^{-3}$ | $4.72 \times 10^{-4}$ | $1.03 \times 10^{-5}$* | $1.85 \times 10^{-7}$* | $1.21 \times 10^{-9}$* | $2.23 \times 10^{-12}$* |
| $CG$ | 2.76 | 4.36 | 5.55 | 6.10 | 6.50 | 6.80 |

In order to give a better idea of the distributions of $N(\boldsymbol{r})$, $C(\boldsymbol{r})$, and $M(\boldsymbol{r})$ for the (72,36) code, the histograms of these random variables corresponding to our experiments are given in figures 1 through 9.

The time and space complexities of Wolf's algorithm [21] are $O(n \times \min(2^k, 2^{n-k}))$ [8] and those of our algorithm are $O(n \times N(\boldsymbol{r}))$ and $O(n \times M(\boldsymbol{r}))$, respectively. Since it is very difficult to determine the "hidden costs" because they depend on a particular implementation, it is fair to compare both algorithms based on the values of $\min(2^k, 2^{n-k}), N(\boldsymbol{r})$, and $M(\boldsymbol{r})$.

For the (72,36) the results given in Table 4 show that our algorithm has reduced dramatically the search space when compared to Wolf's algorithm. In our simulation of $\gamma_b = 2\ dB$, the worst case time (space) complexity of our algorithm is more than 4(5) orders of magnitude smaller than that of Wolf's algorithm. For higher SNR this reduction is even more

drastic. We should also observe that the average values of $N(\boldsymbol{r})$ and $M(\boldsymbol{r})$ are approximately 2 orders of magnitude smaller than those values in the worst case. In summary, the simulation results have far exceeded our most optimistic expectations, especially because quadratic residue codes are known to be very hard to decode even when only hard-decision decoding is employed [4, 7, 5].

In order to verify the contribution of our $h$ function to the efficiency of our decoding algorithm we modified our algorithm by defining another $h$ function which was equal to zero for every node $m$ at level $\ell < k-1$ in the graph, that is, $f(m) = g(m)$. This modified algorithm (MA) is Dijkstra's algorithm [9] combined with most of our speed-up techniques. If Dijkstra's algorithm is applied without using these techniques we expect a much worse performance than that of MA. In Table 7 we give simulation results for the decoding of the $(72, 36)$ code using the proposed algorithm (PA) and MA for $\gamma_b$ equal to 5 $dB$, 6 $dB$, and 7 $dB$. These results were obtained by simulating 100 samples for each SNR.

Table 7: Simulation for the decoding of the (72,36) code using PA and MA

| | $\gamma_b$ | 5 $dB$ | | | 6 $dB$ | | | 7 $dB$ | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | max | min | ave | max | min | ave | max | min | ave |
| PA | $N(\boldsymbol{r})$ | 1345 | 36 | 79 | 631 | 36 | 43 | 89 | 36 | 37 |
| | $C(\boldsymbol{r})$ | 314 | 1 | 10 | 68 | 1 | 3 | 16 | 1 | 2 |
| | $M(\boldsymbol{r})$ | 132 | 36 | 38 | 36 | 36 | 36 | 36 | 36 | 36 |
| MA | $N(\boldsymbol{r})$ | 241634 | 72 | 33358 | 69052 | 72 | 9213 | 15010 | 72 | 2337 |
| | $C(\boldsymbol{r})$ | 43746 | 1 | 5295 | 9442 | 1 | 1173 | 1662 | 1 | 258 |
| | $M(\boldsymbol{r})$ | 17905 | 36 | 2226 | 4382 | 36 | 538 | 724 | 36 | 140 |

Based on the results given in Table 7 we can conclude that our function $h$ contributes drastically to the reduction of search space.

In order to compare fairly paths of different lengths, we again modified PA by reordering list OPEN according to the values of the Fano metric [15] calculated for those nodes opened. The optimality was guaranteed because we deleted only from list OPEN a node $m$ for which

$f(m) > UB$. However, the performance of this modified algorithm was almost identical to that of MA for the samples tried. We conjecture that this fact is due to the reordering of the positions of $\boldsymbol{r}$ according to their reliabilities.

Finally, we give simulation results for the (128,64) binary extended BCH code whose $d_{\min} = 22$. This code is obtained by adding an overall parity check to the (127,64) binary BCH code. As pointed out earlier in the calculation of values of function $h$, the algorithm assumes that the Hamming weights of the codewords of this code belong to the set $\{x|(x$ is even and $22 \leq x \leq 106)$ or $(x = 0)$ or $(x = 128)\}$. Sub-optimal decoding procedures for this code have been proposed in [10, 3].

The simulation results for the (128,64) binary extended BCH code for $\gamma_b$ equal to 5 $dB$, 6 $dB$, 7 $dB$, 8 $dB$, 9 $dB$, and 10 $dB$ are given in the following tables. These results were obtained by simulating 17,000 samples for each SNR. Note that $2^{64} \approx 1.84 \times 10^{19}$.

Table 8: Simulation for the (128,64) code

| $\gamma_b$ | 5 $dB$ | | | 6 $dB$ | | | 7 $dB$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | max | min | ave | max | min | ave | max | min | ave |
| $N(\boldsymbol{r})$ | 524178 | 64 | 1400 | 27345 | 64 | 168 | 2047 | 64 | 71 |
| $C(\boldsymbol{r})$ | 104582 | 1 | 196 | 3150 | 1 | 14 | 144 | 1 | 2 |
| $M(\boldsymbol{r})$ | 44333 | 64 | 136 | 1479 | 64 | 66 | 216 | 64 | 65 |

| $\gamma_b$ | 8 $dB$ | | | 9 $dB$ | | | 10 $dB$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | max | min | ave | max | min | ave | max | min | ave |
| $N(\boldsymbol{r})$ | 560 | 64 | 65 | 65 | 64 | 65 | 65 | 64 | 65 |
| $C(\boldsymbol{r})$ | 62 | 1 | 2 | 2 | 1 | 2 | 2 | 1 | 2 |
| $M(\boldsymbol{r})$ | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 |

Table 9: The number of samples for which ANG($\overrightarrow{\mathbf{0}\boldsymbol{v}_{c^*}}$, $\overrightarrow{\mathbf{0}\boldsymbol{r}^*}$) $\leq \alpha$ holds for the (128,64) code

| $\gamma_b$ | 5 $dB$ | 6 $dB$ | 7 $dB$ | 8 $dB$ | 9 $dB$ | 10 $dB$ |
|---|---|---|---|---|---|---|
| # | 118 | 2176 | 10861 | 16611 | 16998 | 17000 |

Table 10: Bit error probability and coding gain for the (128,64) code

| $\gamma_b$ | 5 $dB$ | 6 $dB$ | 7 $dB$ | 8 $dB$ | 9 $dB$ | 10 $dB$ |
|---|---|---|---|---|---|---|
| $P_b$ | $1.57 \times 10^{-12}$* | $1.71 \times 10^{-16}$* | $1.82 \times 10^{-21}$* | $1.02 \times 10^{-27}$* | $1.43 \times 10^{-35}$* | $1.40 \times 10^{-45}$* |
| $CG$ | 8.85 | 9.22 | 9.50 | 9.70 | 9.85 | 10.00 |

When calculating $P_b$ using (2), the value of $n_d = 243,840$ was taken from [2].

Simulation results for the (128,64) code indicate that a drastic reduction on the search space is achieved for the majority of practical communication systems where the probability of error is less than $10^{-3}$ ($\gamma_b$ greater than 6.8 $dB$) [7] even when the algorithm uses a superset of HW.

# 4 Conclusion

Long linear block codes with coding gains far superior to those of convolutional codes have been known for many years. However, these and other excellent block codes have not been used in practice for lack of an efficient soft-decision decoding algorithm.

In this report we have proposed a novel decoding technique. Simulation results for various linear block codes attest to the fact that this decoding technique drastically reduced the searach space, especially for the majority of practical communication systems where the probability of error is less than $10^{-3}$ ($\gamma_b$ greater than 6.8 $dB$) [7]. Thus, this decoding procedure has not only resulted in an efficient soft-decision decoding algorithm for hitherto intractable linear block codes, but an algorithm which is in fact optimal as well.

Although we have developed our algorithm for linear block codes, there is no reason why we cannot extend this approach to convolutional codes. If the length of the information sequence is very large, we cannot wait for the end of the transmission to start the decoding process as in the case of block codes. In this case, we divide the received vector into suitable blocks of received symbols and apply a modified version of our algorithm to these blocks instead of to the entire received vector as is done in the case of block codes. In this way, we can start the decoding procedure after receiving the first block. We can easily determine the Hamming weights of these blocks when they are part of a codeword. Note that by inspecting these blocks, and not the entire received vector, the algorithm is assuming that they are independent, while actually they are not. Thus, in contrast with what happens with block codes when calculating the value of function $h$ for a node, the algorithm may inspect patterns whose Hamming weights are different from those of any codeword. However, our function $h$ still satisfies $h(m) \leq h^*(m)$ for any node $m$ and thus will always find an optimal solution. Again, because the blocks are not independent, we cannot, within each block, order the received symbols according to the reliability of its positions as is done in the case of block codes. Although we do not yet have simulation results for convolutional codes using our decoding procedure, we predict that the impact will be equally dramatic, and that the use of this decoding approach will make practical, for the first time, optimal soft-decision decoding of convolutional codes with large constraint lengths.

Furthermore, we would like to point out that the algorithm present in this report is suitable for a parallel implementation. One of the reasons is that when calculating $h(m)$ for node $m$, the algorithm has determined the labels of the path from node $m$ to a node at level $k-2$ that it will follow, so the successors of the nodes in this path can be open simultaneously and processed independently. This will reduce substantially the idle time of processors and the overhead due to processor communication. Thus, we expect a very good speed-up from a parallel version of our algorithm.

This decoding approach will impact both the theoretical and practical branches of coding theory. Theoreticians will be challenged to identify and construct classes of linear codes whose properties maximize the efficiency of this decoding procedure. And practitioners will

want to find the most efficient way to implement this algorithm in a fast, single-purpose processor using sequential/parallel structures.


# Acknowledgment

The authors would like to thank Elaine Weinman for her invaluable help in the preparation of this manuscript. Thanks are also due Prof. L. D. Rudolph for his helpful suggestions, and Prof. O. Biham for organizing the Research Experience for Undergraduate Program.


# Appendix A

In this appendix we will give an example to illustrate our claim that Hwang's algorithm [13] has a fallacy. For the following example his algorithm will fail to start.

Consider the (8,4) extended binary Hamming code generated by

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

Let $\boldsymbol{r}$ be the received vector and $\boldsymbol{\phi} = (\phi_0, \phi_1, \ldots, \phi_7)$ the channel measurement information vector of $\boldsymbol{r}$ [13]. Assume that $\phi_0 < 0, \phi_1 < 0, \phi_2 < 0, \phi_3 > 0, \phi_4 > 0, \phi_5 > 0, \phi_6 > 0, \phi_7 > 0$, and

$$|\phi_0| > |\phi_1| > |\phi_2| > |\phi_4| > |\phi_3| > |\phi_5| > |\phi_6| > |\phi_7|.$$

In order to have the first $k$ bits of $\boldsymbol{\phi}$ the most reliable, we must swap positions 3 and 4 in $\boldsymbol{\phi}$ and obtain $\boldsymbol{\phi}' = (\phi_0, \phi_1, \phi_2, \phi_4, \phi_3, \phi_5, \phi_6, \phi_7)$. Corresponding to this exchange we have

$$G' = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

which generates code $C'$.

To start the algorithm we must construct a codeword $\boldsymbol{c}'_1 = (c'_{10}, c'_{11}, \ldots, c'_{17})$ of $C'$ such that

$$(-1)^{c'_{10}} \times \phi_0 > 0, (-1)^{c'_{11}} \times \phi_1 > 0, (-1)^{c'_{12}} \times \phi_2 > 0 \text{ and } (-1)^{c'_{13}} \times \phi_4 > 0.$$

Thus, $c'_{10} = 1, c'_{11} = 1, c'_{12} = 1$, and $c'_{13} = 0$. However, $(1,1,1,1,0,0,0,0)$ and $(1,1,1,1,1,1,1,1)$ are the only codewords in $C'$ whose first three bits are ones. Thus, the algorithm will fail in Step 1. The fallacy is in the assumption that the $k$ most reliable positions of $\boldsymbol{r}$ are linearly independent.

# Appendix B

Let $\boldsymbol{r} = (r_0, r_1, \ldots, r_{n-1})$ be the received vector. Since we are transmitting codewords of $C$ over the **AWGN** channel using antipodal signaling, $r_i$ can be considered to be more reliable than $r_j$ if $|r_i| > |r_j|$ where $|x|$ is the absolute value of $x$ [12]. Let $\boldsymbol{r}' = (r'_0, r'_1, \ldots, r'_{n-1})$ be a vector obtained by permuting the positions of $\boldsymbol{r}$ such that $|r'_i| \geq |r'_{i+1}|$ for $0 \leq i < n-1$. The $k \times n$ matrix $\boldsymbol{G}'$ is obtained from $\boldsymbol{G}$ by applying this same permutation to the columns of $\boldsymbol{G}$. In order to give an algorithm to obtain $\boldsymbol{G}^*$, the generator matrix of $C^*$, from $\boldsymbol{G}'$, we first introduce some definitions.

Let $\boldsymbol{A}$ be an $r \times m$ matrix. Given a set $S = \{i_1, i_2, \ldots, i_s\} \subset \{0, 1, 2, \ldots, m-1\}$ we say that $S$ is a sub-information set of $\boldsymbol{A}$ iff the columns of $\boldsymbol{A}$ indexed by $i_1, i_2, \ldots, i_s$ are linearly independent. Furthermore, we define the $SW$ operator. For $0 \leq i, j < m$, $SW(\boldsymbol{A}, i, j)$ is the $r \times m$ matrix obtained from $\boldsymbol{A}$ by swapping columns $i$ and $j$ of $\boldsymbol{A}$.

The following is an algorithm to obtain $\boldsymbol{G}^*$ from $\boldsymbol{G}'$ for $2 \leq k < n$.

1. $i \leftarrow 1; j \leftarrow 1; S = \{0\}$; $\boldsymbol{G}^*_1 \leftarrow \boldsymbol{G}'$.
2. If $S \cup \{j\}$ is a sub-information set of $\boldsymbol{G}^*_1$, then $\boldsymbol{G}^*_1 \leftarrow SW(\boldsymbol{G}^*_1, i, j)$;
   else
   $j \leftarrow j + 1$;
   go to 2.

3. $S \leftarrow S \cup \{i\}$.

4. If $|S| = k$, then stop;

   else

   $i \leftarrow i + 1$;

   $j \leftarrow j + 1$;

   go to 2.

5. Transform $G_1^*$ into $G^*$ by row operation such that the first $k$ columns of $G^*$ form a $k \times k$ identity matrix.

The time complexity of the procedure to construct $G^*$ is $O(k^2 \times n)$; however, many of the operations performed during this construction can be done in parallel. In this case, the time complexity becomes $O(k \times n)$.

# Appendix C

In this appendix we present an algorithm to calculate $h(m)$ for node $m$ at level $\ell, -1 \leq \ell < k-1$, whose time complexity is $O(n)$. We also show that $T(m) \neq \emptyset$.

Let vector $\boldsymbol{u}_\ell = (u_{\ell 0}, u_{\ell 1}, \ldots, u_{\ell(n-\ell-2)})$ be obtained by permuting the positions of $(r^*_{\ell+1}, r^*_{\ell+2}, \ldots, r^*_{n-1})$ in such a manner that $u_{\ell i} \leq u_{\ell(i+1)}$ for $0 \leq i < n-\ell-2$. Now we define $b \in \{-1, 0, 1, \ldots, n-1\}$ as follows: if $u_{-1(0)} \geq 0$, then $b = -1$; if $u_{-1(n-1)} < 0$, then $b = n-1$; otherwise, $b$ is the position in $\boldsymbol{u}_{-1}$ such that $u_{-1(b)} < 0$ and $u_{-1(b+1)} \geq 0$. Let $\delta = b + \frac{1}{2}$.

Our algorithm computes $h(m)$ using $\boldsymbol{u}_\ell$ instead of $\boldsymbol{r}^*$. This is possible because of the property

$$h(m) = \min_{\boldsymbol{v} \in T(m)} \left\{ \sum_{i=\ell+1}^{n-1} \left( u_{\ell(i-\ell-1)} - (-1)^{v_i} \sqrt{E} \right)^2 \right\}$$

where $T(m)$ is defined in Section 3.2.

This property is easily proved because if $\boldsymbol{v} \in T(m)$, then all binary vectors of the same Hamming weight as $\boldsymbol{v}$ are contained in $T(m)$ and $\boldsymbol{u}_\ell$ is obtained by applying a permutation $\pi_\ell$ to the components of $(r^*_{\ell+1}, r^*_{\ell+2}, \ldots, r^*_{n-1})$. At the end of this appendix we will show how

this permutation can be used to determine the labels of the path with constant value of $f$, which will be the path up to level $k-2$, followed by the decoding algorithm. Furthermore, $\pi_\ell$ can be easily obtained from $\pi_{-1}$.

We now prove some technique lemmas. Consider the set $T_w$ of all binary $(n-\ell-1)$-tuples of Hamming weight $w$. Furthermore, let $\boldsymbol{v}_p = (v_{p0}, v_{p1}, \ldots, v_{p(w-1)}, v_{pw}, \ldots, v_{p(n-\ell-2)}) \in T_w$, where $v_{pi} = 1, 0 \le i < w$ and $v_{pi} = 0, w \le i < n-\ell-1$.

**Lemma C1.** *If* $\boldsymbol{v} = (v_0, v_1, \ldots, v_{n-\ell-2}) \in T_w$, *then*

$$\sum_{i=0}^{n-\ell-2} \left(u_{\ell i} - (-1)^{v_{pi}}\sqrt{E}\right)^2 \le \sum_{i=0}^{n-\ell-2} \left(u_{\ell i} - (-1)^{v_i}\sqrt{E}\right)^2.$$

PROOF.

$$\begin{aligned} D_1 &= \sum_{i=0}^{n-\ell-2} \left(u_{\ell i} - (-1)^{v_{pi}}\sqrt{E}\right)^2 - \sum_{i=0}^{n-\ell-2} \left(u_{\ell i} - (-1)^{v_i}\sqrt{E}\right)^2 \\ &= 2\sqrt{E} \sum_{i=0}^{n-\ell-2} \left\{ u_{\ell_i} \left((-1)^{v_i} - (-1)^{v_{pi}}\right) \right\} \end{aligned}$$

Let $S = \{x | v_x = 0 \text{ and } 0 \le x < w\}$ and $S' = \{x | v_x = 1 \text{ and } w \le x < n-\ell-1\}$. Since $\boldsymbol{v}_p = (1, 1, \ldots, 1, 0, 0, \ldots, 0)$ and $W_H(\boldsymbol{v}) = W_H(\boldsymbol{v}_p)$, then $|S| = |S'|$. So $D_1 = 4\sqrt{E}(\sum_{i \in S} u_{\ell i} - \sum_{i \in S'} u_{\ell i}) \le 0$ since $|S| = |S'|$ and $u_{\ell i} \le u_{\ell j}$, $i \in S$ and $j \in S'$. □

Let $S_\ell = \{x | u_{\ell x} < 0\}$ and $\boldsymbol{v}'_p = (v'_{p0}, v'_{p1}, \ldots, v'_{p(w'-1)}, v'_{pw'}, \ldots, v'_{p(n-\ell-2)}) \in T_{w'}$, where $v'_{pi} = 1$, $0 \le i < w'$ and $v'_{pi} = 0$, $w' \le i < n-\ell-1$.

**Lemma C2.** *If* $w' < w \le |S_\ell|$, *then*

$$\sum_{i=0}^{n-\ell-2} \left(u_{\ell i} - (-1)^{v_{pi}}\sqrt{E}\right)^2 < \sum_{i=0}^{n-\ell-2} \left(u_{\ell i} - (-1)^{v'_{pi}}\sqrt{E}\right)^2.$$

PROOF.

$$\begin{aligned} D_2 &= \sum_{i=0}^{n-\ell-2} \left(u_{\ell i} - (-1)^{v_{pi}}\sqrt{E}\right)^2 - \sum_{i=0}^{n-\ell-2} \left(u_{\ell i} - (-1)^{v'_{pi}}\sqrt{E}\right)^2 \\ &= 4\sqrt{E}\sum_{i=w'}^{w-1} u_{\ell i} < 0 \text{ since } u_{\ell i} < 0, 0 \le i < w. \end{aligned}$$

□

**Lemma C3.** *If* $|S_\ell| < w < w'$, *then*

$$\sum_{i=0}^{n-\ell-2} \left(u_{\ell i} - (-1)^{v_{pi}}\sqrt{E}\right)^2 \le \sum_{i=0}^{n-\ell-2} \left(u_{\ell i} - (-1)^{v'_{pi}}\sqrt{E}\right)^2.$$

The proof of this lemma is similar to that in Lemma C2.

Let $c_0^*, c_1^*, \ldots, c_\ell^*$ be the labels of the path $P'_m$ from the start node to node $m$ at level $\ell$ found so far by the decoding algorithm. Furthermore, let $c_{\ell+1}^*, c_{\ell+2}^*, \ldots, c_{n-1}^*$ be the labels of a path $\boldsymbol{P}_m$ from node $m$ at level $\ell$ to the goal node. Note that $W(\boldsymbol{P}_m)$ can only have values that belong to the set $Q = \{w_i - W(\boldsymbol{P}'_m) | 0 \le w_i - W(\boldsymbol{P}'_m) \le n - \ell - 1 \text{ and } 0 \le i \le I\}$.

We now show that $Q \neq \emptyset$ and thus $T(m) \neq \emptyset$. Let $c_0^*, c_1^*, \ldots c_\ell^*$ be the labels of $\boldsymbol{P}'_m$. Furthermore, let $\boldsymbol{u} = (c_0^*, c_1^*, \ldots c_\ell^*, 0, 0, \ldots, 0)$ be a binary $k$-tuple. Then $\boldsymbol{u} \cdot \boldsymbol{G}^*$ is a codeword in $\boldsymbol{C}^*$. Since the first $k$ columns of $\boldsymbol{G}^*$ form a $k \times k$ identity matrix and $\ell < k - 1$, then $(W_H(\boldsymbol{u} \cdot \boldsymbol{G}^*) - W(\boldsymbol{P}'_m)) \in Q$.

Let $J \in \{0, 1, \ldots, I\}$ such that $w_J - W(\boldsymbol{P}'_m)$ is the smallest value in $Q$. Analogously, let $I' \in \{0, 1, \ldots, I\}$ such that $w_{I'} - W(\boldsymbol{P}'_m)$ is the largest value in $Q$.

By Lemma C1, our algorithm to compute $h(m)$ only needs to consider vectors of the form $\boldsymbol{v}_p = (v_{p0}, v_{p1}, \ldots, v_{p(n-\ell-2)}) = (1, 1, \ldots, 1, 0, 0, \ldots, 0)$ with Hamming weights $w_i - W(\boldsymbol{P}'_m), J \le i \le I'$. Furthermore, by lemmas C2 and C3, we only need to consider the following cases:

*Case 1.* $|S_\ell| < (w_J - W(\boldsymbol{P}'_m))$. So,

$$h(m) = \sum_{i=0}^{n-\ell-2} \left(u_{\ell i} - (-1)^{v_{pi}}\sqrt{E}\right)^2,$$

where $W_H(\boldsymbol{v}_p) = w_J - W(\boldsymbol{P}'_m)$.

*Case 2.* $|S_\ell| \geq (w_{I'} - W(\boldsymbol{P}'_m))$. So,

$$h(m) = \sum_{i=0}^{n-\ell-2} \left(u_{\ell i} - (-1)^{v_{pi}}\sqrt{E}\right)^2,$$

where $W_H(\boldsymbol{v}_p) = w_{I'} - W(\boldsymbol{P}'_m)$.

*Case 3.* $w_{i_1} - W(\boldsymbol{P}'_m) \leq |S_\ell| < w_{i_1+1} - W(\boldsymbol{P}'_m)$. So, $h(m) = \min\{A_1, A_2\}$, where $A_1 = \sum_{i=0}^{n-\ell-2} \left(u_{\ell i} - (-1)^{v_{pi}}\sqrt{E}\right)^2$ and $W_H(\boldsymbol{v}_p) = w_{i_1} - W(\boldsymbol{P}'_m)$, and $A_2 = \sum_{i=0}^{n-\ell-2} \left(u_{\ell i} - (-1)^{v_{pi}}\sqrt{E}\right)^2$ and $W_H(\boldsymbol{v}_p) = w_{i_1+1} - W(\boldsymbol{P}'_m)$.

Thus, given $\boldsymbol{u}_\ell$ and $|S_\ell|$, the time complexity of computing $h(m)$ is $O(n)$. However, we can by simple inspection construct $\boldsymbol{u}_\ell$ and compute $|S_\ell|$ from $\boldsymbol{u}_{-1}$ and $\delta$. So, the time complexity of the entire procedure to compute $h(m)$ is $O(n)$.

We now give a procedure to construct the path $\overline{\boldsymbol{P}}_m = (m_\ell, m_{\ell+1}, \ldots, m_{\ell+(k-\ell-2)})$ with constant value of $f$ which will be the path up to level $k-2$ followed by the algorithm from node $m_\ell = m$. In Appendix E we show that $f(m_{\ell+i}) = f(m_\ell)$ for $i = 1, 2, \ldots, (k-\ell-2)$.

Let $\boldsymbol{v}_p = (v_{p0}, v_{p1}, \ldots, v_{p(n-\ell-2)}) \in T(m)$ be the vector used to calculate $h(m)$. Furthermore, define the function $t$ from $\{\ell+1, \ell+2, \ldots, n-1\}$ to $\{0, 1, \ldots, n-\ell-2\}$ as

$$t(\pi_\ell(\ell+1+i)) = i.$$

Now the labels of path $\overline{\boldsymbol{P}}_m$ are given by

$$v_{pt(\ell+1)}, v_{pt(\ell+2)}, \ldots, v_{pt(\ell+(k-\ell-2))}.$$

Thus, $\overline{\boldsymbol{P}}_m$ can be determined from these labels.

# Appendix D

Let node $m_2$ at level $\ell$ be a successor of node $m_1$. Furthermore, let $c^*_\ell$ be the label of the arc from node $m_1$ to node $m_2$ and $c(m_1, m_2) = \left(r^*_\ell - (-1)^{c^*_\ell}\sqrt{E}\right)^2$. We now prove that $h(m_1) \leq h(m_2) + c(m_1, m_2)$.

(a) $\ell < k-1$. Let $\boldsymbol{v}= (v_{\ell+1}, v_{\ell+2}, \ldots, v_{n-1}) \in T(m_2)$ such that $h(m_2) = \sum_{i=\ell+1}^{n-1} \left(r_i^* - (-1)^{v_i}\sqrt{E}\right)^2$. Since $\boldsymbol{v} \in T(m_2)$, then $(c_\ell^*, v_{\ell+1}, v_{\ell+2}, \ldots, v_{n-1}) \in T(m_1)$. Thus $\sum_{i=\ell+1}^{n-1} \left(r_i^* - (-1)^{v_i}\sqrt{E}\right)^2 + c(m_1, m_2) \geq h(m_1)$, i.e., $h(m_2) + c(m_1, m_2) \geq h(m_1)$.

(b) $\ell = k-1$. $h(m_1) \leq h^*(m_1)$ and $h(m_2) = h^*(m_2)$. Since $h^*(m_1) - c(m_1, m_2) \leq h^*(m_2)$, then $h(m_1) \leq h^*(m_2) + c(m_1, m_2) = h(m_2) + c(m_1, m_2)$.

(c) $\ell > k-1$. $h(m_1) = h^*(m_1)$ and $h(m_2) = h^*(m_2)$. Since $h^*(m_1) - c(m_1, m_2) = h^*(m_2)$, then $h(m_1) = h(m_2) + c(m_1, m_2)$.

# Appendix E

Consider node $m$ at level $\ell$, $-1 \leq \ell < k-2$. Furthermore, let $h(m) = \sum_{j=1}^{n-\ell-1} \left(r_{\ell+j}^* - (-1)^{v_{\ell+j}}\sqrt{E}\right)^2$. Now consider the path $\overline{P}_{m_\ell} = (m_\ell, m_{\ell+1}, \ldots, m_{\ell+(k-\ell-2)})$ from node $m_\ell = m$ to node $m_{\ell+(k-\ell-2)}$ at level $k-2$ whose labels are $v_{\ell+1}, v_{\ell+2}, \ldots, v_{\ell+(k-\ell-2)}$. We now show by contradiction that if $m_{\ell+1}$ is a node in this path at level $\ell+1$, then $f(m_\ell) = f(m_{\ell+1})$.

Assume $f(m_\ell) \neq f(m_{\ell+1})$. By definition $f(m_\ell) = g(m_\ell)+h(m_\ell)$ and $f(m_{\ell+1}) = g(m_{\ell+1})+ h(m_{\ell+1}) = g(m_\ell) + c(m_\ell, m_{\ell+1}) + h(m_{\ell+1})$ where $c(m_\ell, m_{\ell+1}) = \left(r_{\ell+1}^* - (-1)^{v_{\ell+1}}\sqrt{E}\right)^2$. Thus, $h(m_\ell) \neq c(m_\ell, m_{\ell+1}) + h(m_{\ell+1})$. Since $h(m_\ell) = \sum_{j=1}^{n-\ell-1} \left(r_{\ell+j}^* - (-1)^{v_{\ell+j}}\sqrt{E}\right)^2$, thus $\sum_{j=2}^{n-\ell-1} \left(r_{\ell+j}^* - (-1)^{v_{\ell+j}}\sqrt{E}\right)^2 \neq h(m_{\ell+1})$. Since $(v_{\ell+2}, v_{\ell+3}, \ldots, v_{\ell+(n-\ell-1)}) \in T(m_{\ell+1})$, then $\sum_{j=2}^{n-\ell-1} \left(r_{\ell+j}^* - (-1)^{v_{\ell+j}}\sqrt{E}\right)^2 > h(m_{\ell+1})$. Thus $h(m_\ell) > c(m_\ell, m_{\ell+1}) + h(m_{\ell+1})$. But by the result in Appendix D, $h(m_\ell) \leq c(m_\ell, m_{\ell+1}) + h(m_{\ell+1})$, which is a contradiction. The same argument can be used to prove that $f(m_{\ell+2}), \ldots, f(m_{\ell+(k-\ell-2)})$ are all equal to $f(m_\ell)$, since the $f$ values of the sequence of nodes expanded by our algorithm are nondecreasing. Note that for any node $m$, successor of $m_{\ell+(k-\ell-2)}$, $f(m) = g(m) + h^*(m)$.

We remark here that the path $\overline{\boldsymbol{P}}_{m_\ell}$ is constructed when $h(m_\ell)$ is calculated (see Appendix C).

# Appendix F

In this appendix we give an outline of our decoding algorithm. Recall that we do not check for repeated nodes, thus we do not have to store list CLOSED. We also give the orders of time and space complexities of this algorithm.

Given $\boldsymbol{r}$:

1. $\boldsymbol{r} \longleftarrow \beta\boldsymbol{r}$ where $\beta\boldsymbol{r}$ lies on the surface of the sphere of radius $\sqrt{nE}$ centered at $\mathbf{0}$.
2. Construct $\boldsymbol{G}^*$ and $\boldsymbol{r}^*$ (see Appendix B); store the permutation used to construct $\boldsymbol{r}^*$ from $\boldsymbol{r}$.
3. Construct $\boldsymbol{u}_{-1}$ and find $\pi_{-1}$ and $\delta$ (see Appendix C).
4. Calculate $\alpha$.
5. $|S_{-1}| \longleftarrow \lceil \delta \rceil$; calculate $f(s) = h(s)$ and construct $\overline{\boldsymbol{P}}_s$ (see Appendix C).
6. RESULT_$f \longleftarrow 4nE$; RESULT_$c^* \longleftarrow \mathbf{0}$.
7. Create OPEN containing only the start node.
8. LOOP: Select a node on OPEN with minimum value $f$, remove it from OPEN. Call this node $m$. (Note that whenever node $m_2$ is inserted into OPEN in Step 12(b), then this node can always be selected as node $m$.)
9. If the level of node $m$ is $k-1$, then go to Step 14.
10. Expand node $m$, generating nodes $m_1$ and $m_2$, the successors of node $m$.
11. If $m$ is at level $k-2$, then for $i = 1$ to 2

(a) Construct the codeword $\boldsymbol{c}^*$ whose information bits are given by the labels of the path from the start node to node $m_i$.

(b) Calculate $g(m_i)$ and $h(m_i)$:

$$f(m_i) \longleftarrow g(m_i) + h(m_i).$$

(c) If ANG($\overrightarrow{\mathbf{0}\boldsymbol{v}_{c^*}}$, $\overrightarrow{\mathbf{0}\boldsymbol{r}^*}$) $\leq \alpha$, then

RESULT_$c^*$ $\longleftarrow$ $\boldsymbol{c}^*$;

goto Step 14.

(d) If $f(m_i) <$ RESULT_$f$, then

RESULT_$c^*$ $\longleftarrow$ $\boldsymbol{c}^*$;

RESULT_$f$ $\longleftarrow$ $f(m_i)$;

remove all nodes on OPEN

whose $f$ values are equal to or

greater than RESULT_$f$;

insert $m_i$ into OPEN.

(e) If $f(m_i) \geq$ RESULT_$f$, then discard node $m_i$.

12. If $m$ is at level $\ell < k-2$, then

(a) Select the node that is not on $\overline{\boldsymbol{P}}_m$,

call it $m_1$;

Calculate $g(m_1)$ and $h(m_1)$;

If $\ell < k-3$ then construct $\overline{\boldsymbol{P}}_{m_1}$ (see Appendix C);

$f(m_1) \longleftarrow g(m_1) + h(m_1)$;

If $f(m_1) <$ RESULT_$f$ then

insert node $m_1$ into OPEN;

otherwise discard node $m_1$.

(b) $f(m_2) \longleftarrow f(m)$;

If $\ell < k-3$, then construct $\overline{\boldsymbol{P}}_{m_2}$ by deleting node $m$ from $\overline{\boldsymbol{P}}_m$;

Insert $m_2$ into OPEN as first node.

13. Go LOOP.

14. Construct $\hat{c}$ from RESULT_$c^*$ by applying the inverse permutation that was used to construct $\boldsymbol{r}^*$ from $\boldsymbol{r}$.

We now apply the algorithm to a numerical example. In order for this example to be more illustrative we do not calculate $\alpha$ and ANG($\overrightarrow{\mathbf{0}\boldsymbol{v}_{c^*}}$, $\overrightarrow{\mathbf{0}\boldsymbol{r}^*}$) so we do not perform steps 1, 4, and 11 (c) of the algorithm.

**Example:** Let $\boldsymbol{C}$ be the (8,4) binary extended Hamming code with $w_0 = 0$, $w_1 = 4$, and $w_2 = 8$. $\boldsymbol{C}$ is transmitted over an **AWGN** channel with $E = 1$. The generator matrix of $\boldsymbol{C}$ is

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

Assume that the received vector is $\boldsymbol{r} = (-3, -2, -2, 1, 4, -1, 0, 0)$. We identify a node $m$ at level $\ell$, $0 \leq \ell \leq k-1$ in the tree by $m(c_0^*, c_1^*, \ldots, c_\ell^*)$, where $c_0^*, c_1^*, \ldots, c_\ell^*$ are the labels associated with the path $\boldsymbol{P}'_m$ from start node to node $m$.

**STEP 2:**

$$\boldsymbol{r}' = (4, -3, -2, -2, 1, -1, 0, 0),$$

$$\boldsymbol{G}' = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix},$$

$$\boldsymbol{G}_1^* = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix},$$

$$G^* = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix},$$

$$\boldsymbol{r}^* = (4, -3, -2, 1, -2, -1, 0, 0),$$

and $\pi = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 2 & 4 & 3 & 0 & 5 & 6 & 7 \end{pmatrix}$, where $\pi$ is a permutation used to construct $\boldsymbol{r}^*$ from $\boldsymbol{r}$.

**STEP 3:**

$$\boldsymbol{u}_{-1} = (-3, -2, -2, -1, 0, 0, 1, 4),$$

$$\pi_{-1} = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 7 & 0 & 1 & 6 & 2 & 3 & 4 & 5 \end{pmatrix},$$

and $\delta = 3.5$.

**STEP 5:**

$|S_{-1}| = 4$. Since $|S_{-1}| = 4$ and $W(\boldsymbol{P}'_s) = 0$, then the conditions of Case 3 in Appendix C are satisfied for $w_{i_1} = 4$ and $w_{i_1+1} = 8$. Thus, $h(s) = f(s) = \min\{A_1, A_2\}$, where

$$\begin{aligned} A_1 &= (-3+1)^2 + (-2+1)^2 + (-2+1)^2 + (-1+1)^2 + (0-1)^2 + (0-1)^2 \\ &\quad + (1-1)^2 + (4-1)^2 = 17, \\ A_2 &= (-3+1)^2 + (-2+1)^2 + (-2+1)^2 + (-1+1)^2 + (0+1)^2 + (0+1)^2 \\ &\quad + (1+1)^2 + (4+1)^2 = 37. \end{aligned}$$

Since $A_1 < A_2$, then $\boldsymbol{v}_p = (1, 1, 1, 1, 0, 0, 0, 0)$. Using $(\pi_{-1})^{-1}$ we obtain the labels of $\overline{\boldsymbol{P}}_s$ as <0, 1, 1>.

**STEP 6:**

$$\text{RESULT\_}f = 8 + \sum_{i=0}^{7} r_i^2 = 43,$$

and RESULT_$\boldsymbol{c}^* = (0, 0, 0, 0, 0, .0, 0, 0)$.

**STEP 7**:

OPEN = $<s>$.

**LOOP 1**:

**STEP 8**: Consider node $s$.

**STEP 10**: Expand node $s$ to obtain $m(0)$ and $m(1)$.

**STEP 12(a)**:

$m_1 = m(1)$. Since $\boldsymbol{u}_0 = (-3,-2,-2,-1,0,0,1)$, $|S_0| = 4$ and $W(\boldsymbol{P}'_{m_1}) = 1$, then the conditions of Case 3 in Appendix C are satisfied for $w_{i_1} = 4$ and $w_{i_1+1} = 8$. Thus $h(m_1) = \min\{A_1, A_2\}$, where

$$A_1 = (-3+1)^2 + (-2+1)^2 + (-2+1)^2 + (-1-1)^2 + (0-1)^2 + (0-1)^2 + (1-1)^2 = 12,$$
$$A_2 = (-3+1)^2 + (-2+1)^2 + (-2+1)^2 + (-1+1)^2 + (0+1)^2 + (0+1)^2 = 12.$$

Since $A_1 \leq A_2$, then $\boldsymbol{v}_p = (1,1,1,0,0,0)$. Using $(\pi_{-1})^{-1}$ and the fact that $\boldsymbol{u}_0$ is constructed from $\boldsymbol{u}_{-1}$ by deleting its last component, we obtain the labels of $\overline{\boldsymbol{P}}_{m_1}$ as $<1,1>$. $g(m_1) = (4+1)^2 = 25$, so $f(m_1) = 25 + 12 = 37$. Since $37 < 43$, then OPEN = $<m(1)>$,

**STEP 12(b)**:

$\overline{\boldsymbol{P}}_{m(0)} = <1,1>$ and $f(m(0)) = 17$.

OPEN = $<m(0), m(1)>$.

**LOOP 2**:

**STEP 8**: Consider node $m(0)$.

**STEP 10**: Expand node $m(0)$ to obtain $m(0,0)$ and $m(0,1)$.

**STEP 12(a)**:

$m_1 = m(0,0)$. Since $\boldsymbol{u}_1 = (-2,-2,-1,0,0,1)$, $|S_1| = 3$ and $W(\boldsymbol{P}'_{m_1}) = 0$, then the conditions of Case 3 in Appendix C are satisfied for $w_{i_1} = 0$ and $w_{i_1+1} = 4$. Thus $h(m_1) = \min\{A_1, A_2\}$, where

$$A_1 = (-2-1)^2 + (-2-1)^2 + (-1-1)^2 + (0-1)^2 + (0-1)^2 + (1-1)^2 = 24,$$
$$A_2 = (-2+1)^2 + (-2+1)^2 + (-1+1)^2 + (0+1)^2 + (0-1)^2 + (1-1)^2 = 4.$$

Since $A_2 < A_1$, then $\boldsymbol{v}_p = (1,1,1,1,0,0)$. Using $(\pi_{-1})^{-1}$ and the fact that $\boldsymbol{u}_1$ is constructed from $\boldsymbol{u}_{-1}$ by deleting its last and first components, we obtain the label of $\overline{\boldsymbol{P}}_{m_1}$ as <1>. $g(m_1) = (4-1)^2+(-3-1)^2 = 25$, so $f(m_1) = 25+4 = 29$. Since $29 < 43$, then OPEN = $<m(0,0), m(1)>$.

**STEP 12(b):**

$$\overline{\boldsymbol{P}}_{m(0,1)} = <1> \text{ and f(m(0,1))=17.}$$
$$\text{OPEN } = <m(0,1), m(0,0), m(1)>.$$

**LOOP 3:**

**STEP 8:** Consider node $m(0,1)$.

**STEP 10:** Expand node $m(0,1)$ to obtain $m(0,1,0)$ and $m(0,1,1)$.

**STEP 12(a):**

$m_1 = m(0,1,0)$. Since $\boldsymbol{u}_2 = (-2,-1,0,0,1)$, $|S_2| = 2$ and $W(\boldsymbol{P}'_{m_1}) = 1$, then the conditions of Case 1 in Appendix C is satisfied for $W_J = 4$. Thus $h(m_1) = (-2+1)^2 + (-1+1)^2 + (0+1)^2 + (0-1)^2 + (1-1)^2 = 3$. $g(m_1) = (4-1)^2 + (-3+1)^2 + (-2-1)^2 = 22$, so $f(m_1) = 22+3 = 25$. Since $25 < 43$, then OPEN = $<m(0,1,0), m(0,0), m(1)>$.

**STEP 12(b):**

$$f(m(0,1,1)) = 17.$$
$$\text{OPEN } = <m(0,1,1), m(0,1,0), m(0,0), m(1)>.$$

**LOOP 4:**

**STEP 8:** Consider node $m(0,1,1)$.

**STEP 10:** Expand node $m(0,1,1)$ to obtain $m_1 = m(0,1,1,0)$ and $m_2 = m(0,1,1,1)$.

**STEP 11(a):** Pick $m_1$;

$$\boldsymbol{c}^* = (0,1,1,0,0,0,1,1).$$

**STEP 11(b):**

$$f(m_1) = (4-1)^2 + (-3+1)^2 + (-2+1)^2 + (1-1)^2 + (-2-1)^2$$
$$+(-1-1)^2 + (0+1)^2 + (0+1)^2 = 29.$$

**STEP 11(d):**

RESULT_$c^*$ = $(0,1,1,0,0,0,1,1)$.

RESULT_$f$ = 29.

OPEN = $<m(0,1,0), m(0,1,1,0)>$.

**STEP 11(a):**Pick $m_2$;

$\boldsymbol{c}^* = (0,1,1,1,0,1,0,0)$.

**STEP 11(b):**

$$f(m_2) = (4-1)^2 + (-3+1)^2 + (-2+1)^2 + (1+1)^2$$
$$+(-2-1)^2 + (-1+1)^2 + (0-1)^2 + (0-1)^2 = 29.$$

**STEP 11(e):** discard node $m_2 = m(0,1,1,1)$.

**LOOP 5:**

**STEP 8:** Consider node $m(0,1,0)$.

**STEP 10:** Expand node $m(0,1,0)$ to obtain $m_1 = m(0,1,0,0)$ and $m_2 = m(0,1,0,1)$.

**STEP 11(a):** Pick $m_1$;

$$\boldsymbol{c}^* = (0,1,0,0,1,1,0,1).$$

**STEP 11(b):**

$$f(m_1) = (4-1)^2 + (-3+1)^2 + (-2-1)^2 + (1-1)^2$$
$$+(-2+1)^2 + (-1+1)^2 + (0-1)^2 + (0+1)^2 = 25.$$

**STEP 11(d):**

RESULT_$c^*$ = $(0,1,0,0,1,1,0,1)$.

RESULT_$f = 25$.

OPEN = $<m(0,1,0,0)>$.

**STEP 11(a):** Pick $m_2$; $\quad c^* = (0,1,0,1,1,0,1,0)$.

**STEP 11(b):**

$$\begin{aligned} f(m_2) &= (4-1)^2 + (-3+1)^2 + (-2-1)^2 + (1+1)^2 \\ &\quad +(-2+1)^2 + (-1-1)^2 + (0+1)^2 + (0-1)^2 = 33. \end{aligned}$$

**STEP 11(e):** Discard $m_2 = m(0,1,0,1)$.

**LOOP 6:**

**STEP 8:** Consider node $m(0,1,0,0)$.

**STEP 9:** Go to STEP 14.

**STEP 14:**

$$\hat{c} = (1,0,1,0,0,1,0,1).$$

In Table F1 we give the orders of time and space complexities for each step of the algorithm. One way of implementing list OPEN is to use a $B$-tree [19]. The time complexities of steps 8, 11(d), and 12(a) given in Table F1 assume that this data structure is used. These complexities are obtained by noticing that the maximum number of nodes visited during decoding of $\boldsymbol{r}$ are upperbounded by $2^{k+1} - 1$, the number of nodes in a complete binary tree of height $k$. Thus, the time complexity of step 8 is $O(k)$, and that of steps 11(d) and 12(a) is $O(n)$.

We remark here that Step 11(a) is not performed for all the nodes visited (open) during the decoding procedure. It is performed only for those nodes visited at level $k-1$. Furthermore, in this step some operations performed during the construction of a codeword $\boldsymbol{u} \cdot \boldsymbol{G}^*$ can be done in parallel. So, the time complexity of this step becomes $O(n)$ which we will assume to be the case in the following analysis.

Table F1: Order of Complexities

| Complexities | | |
|---|---|---|
| Step | Time | Space |
| 1 | $O(n)$ | $O(n)$ |
| 2 | $O(k \times n)$ | $O(k \times n)$ |
| 3 | $O(n \times \log n)$ | $O(n)$ |
| 4 | $O(n)$ | $O(1)$ |
| 5 | $O(n)$ | $O(n)$ |
| 6 | $O(n)$ | $O(n)$ |
| 7 | $O(1)$ | $O(n)$ |
| 8 | $O(\log(2^{k+1}-1))$ | $O(n)$ |
| 9 | $O(1)$ | $O(1)$ |
| 10 | $O(k)$ | $O(n)$ |
| 11(a) | $O(k \times n)$ | $O(n)$ |
| 11(b) | $O(n)$ | $O(1)$ |
| 11(c) | $O(n)$ | $O(1)$ |
| 11(d) | $O(n+\log(2^{k+1}-1))$ | $O(n)$ |
| 11(e) | $O(1)$ | $O(1)$ |
| 12(a) | $O(n+\log(2^{k+1}-1))$ | $O(1)$ |
| 12(b) | $O(1)$ | $O(1)$ |
| 13 | $O(1)$ | $O(1)$ |
| 14 | $O(n)$ | $O(1)$ |

In order to give the time and space complexities of our algorithm, we define the following quantities:

$N(\boldsymbol{r})$ = the number of nodes visited during the decoding of $\boldsymbol{r}$;

$M(\boldsymbol{r})$ = maximum |OPEN| that will occur during the decoding of $\boldsymbol{r}$, where |OPEN| denotes the number of nodes on list OPEN.

Based on the information given in Table F1, we have

(a) time complexity is $O(k \times n + n \times N(\boldsymbol{r}))$;

(b) space complexity is $O(k \times n + n \times M(\boldsymbol{r}))$.

Since a lower bound for $N(\boldsymbol{r})$ and $M(\boldsymbol{r})$ is $k$, then we may write the time complexity as $O(n \times N(\boldsymbol{r}))$, and the space complexity as $O(n \times M(\boldsymbol{r}))$.

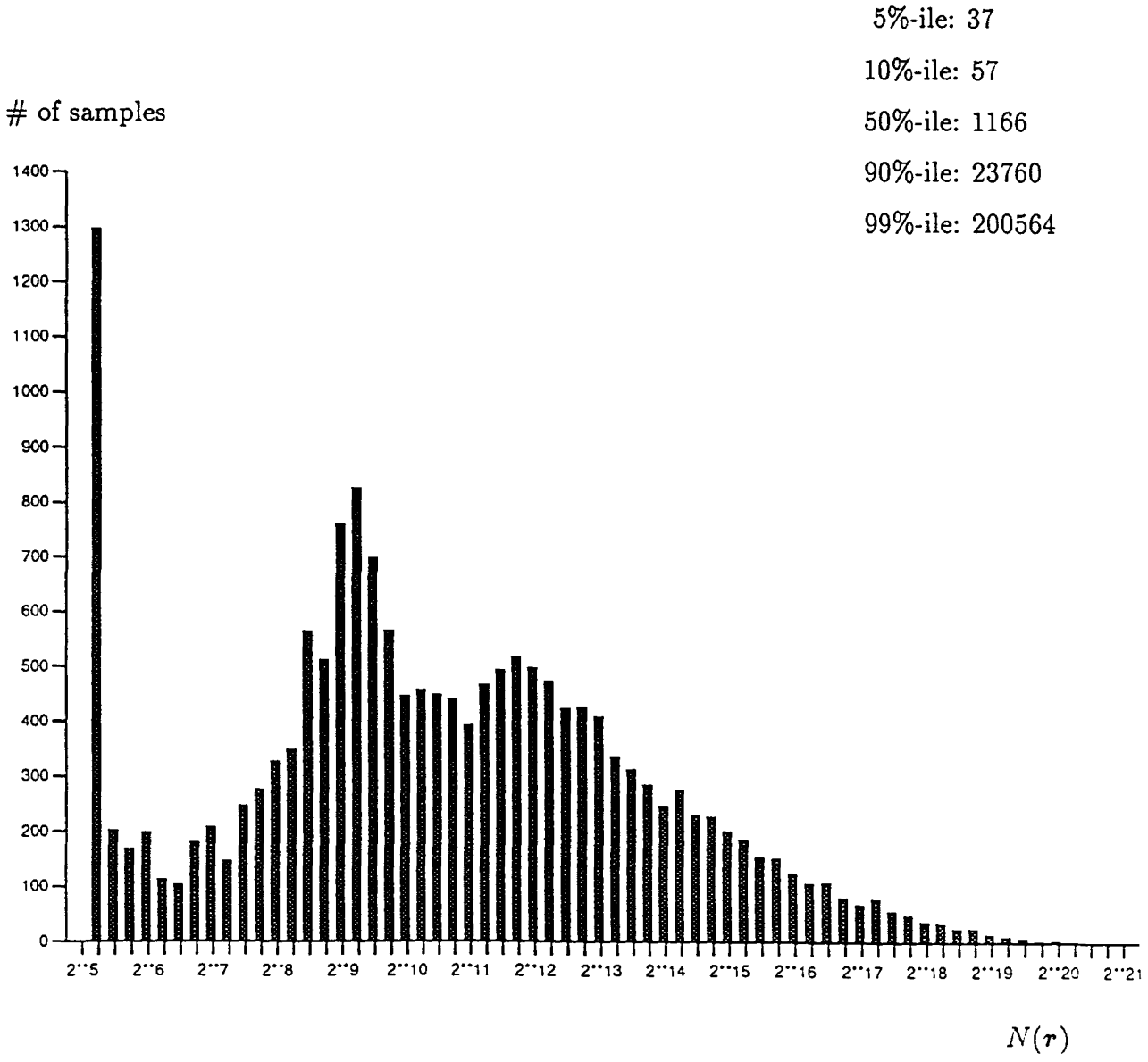


Fig. 1. Histogram representing $N(\boldsymbol{r})$ for 17,000 received vectors obtained by transmitting the (72,36) code over AWGN channel at $\gamma_b = 2\ dB$.

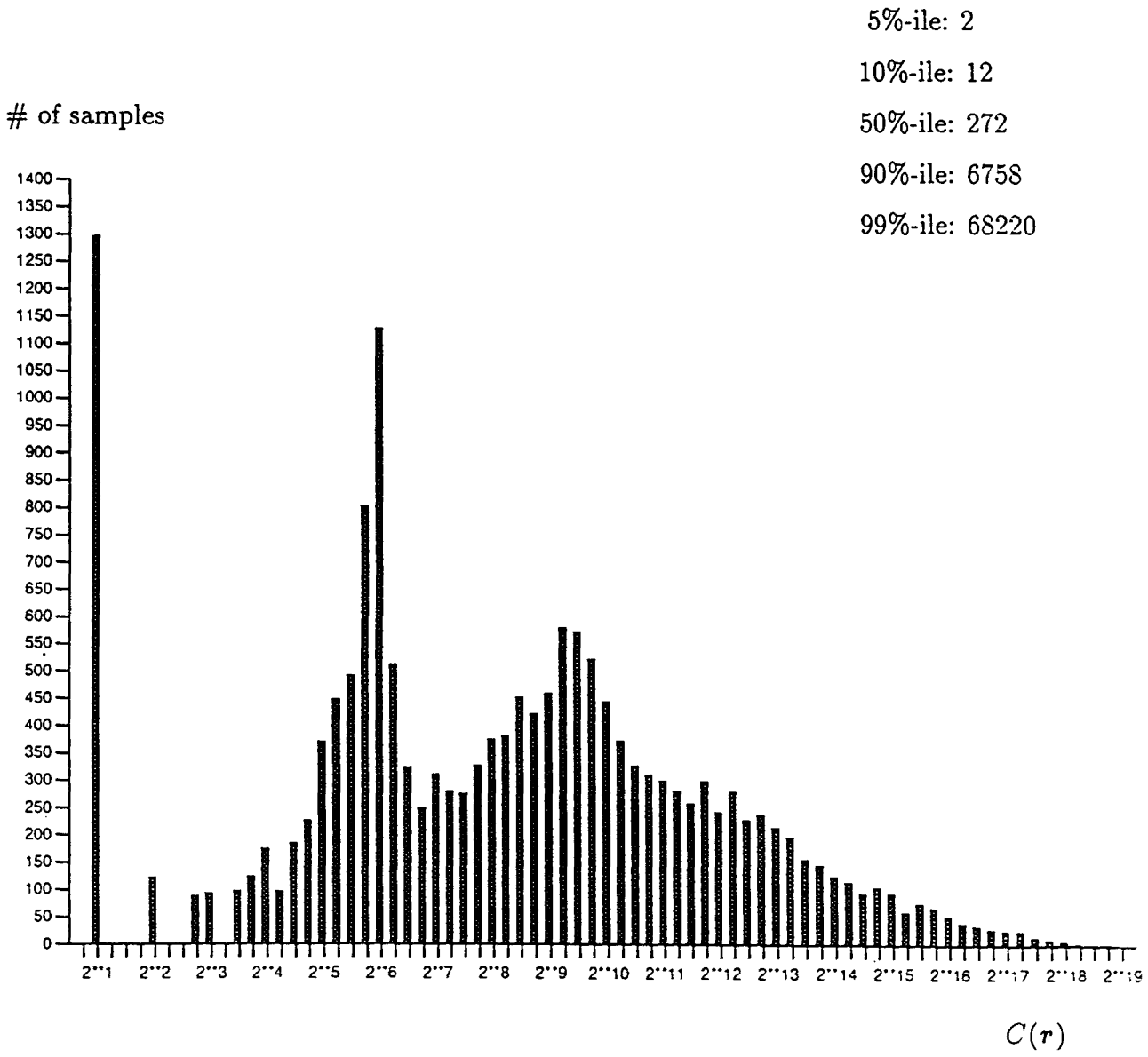


Fig. 2. Histogram representing $C(\boldsymbol{r})$ for 17,000 received vectors obtained by transmitting the (72,36) code over AWGN channel at $\gamma_b = 2\ dB$.

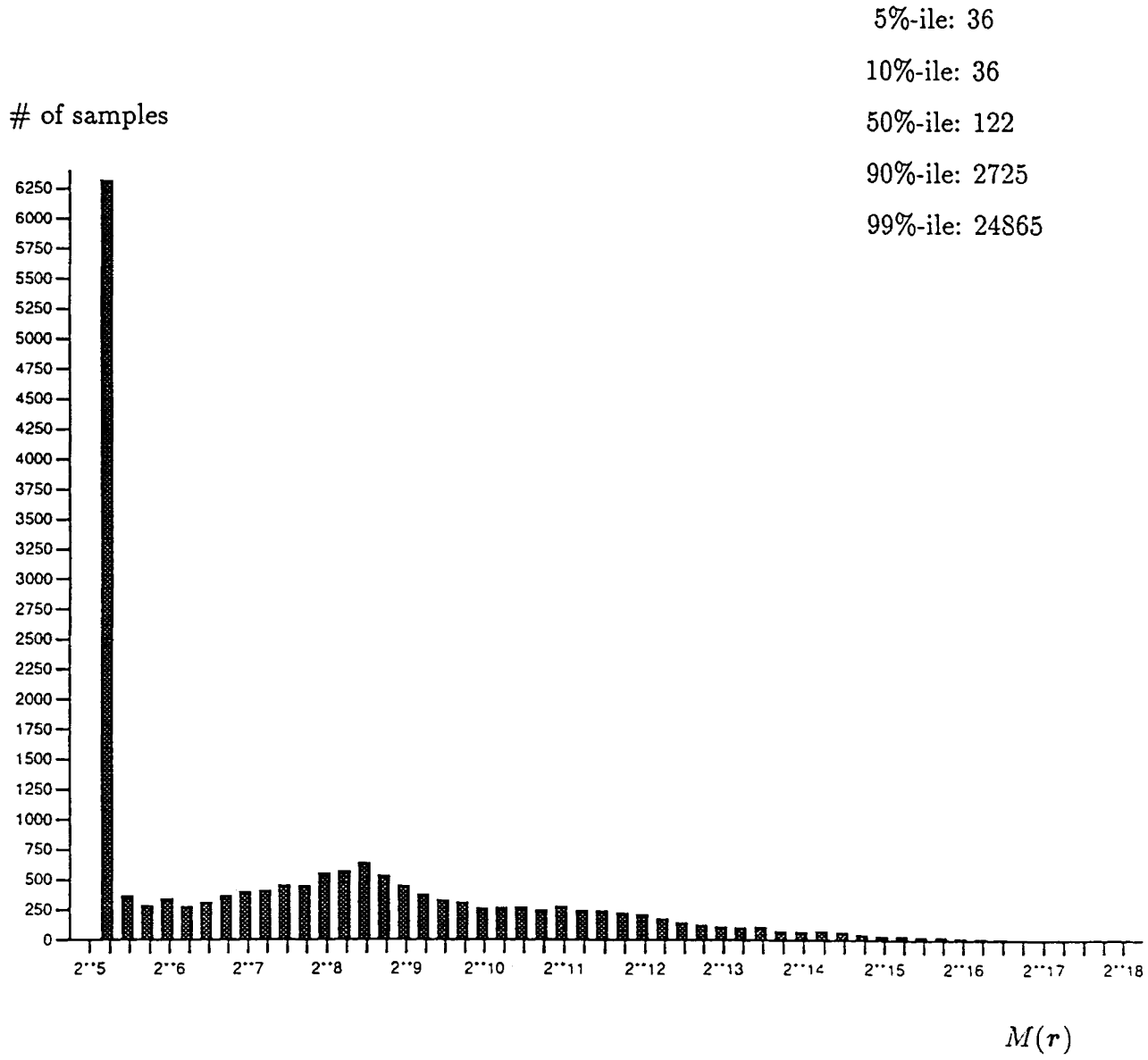


Fig. 3. Histogram representing $M(\boldsymbol{r})$ for 17,000 received vectors obtained by transmitting the (72,36) code over AWGN channel at $\gamma_b = 2\ dB$.

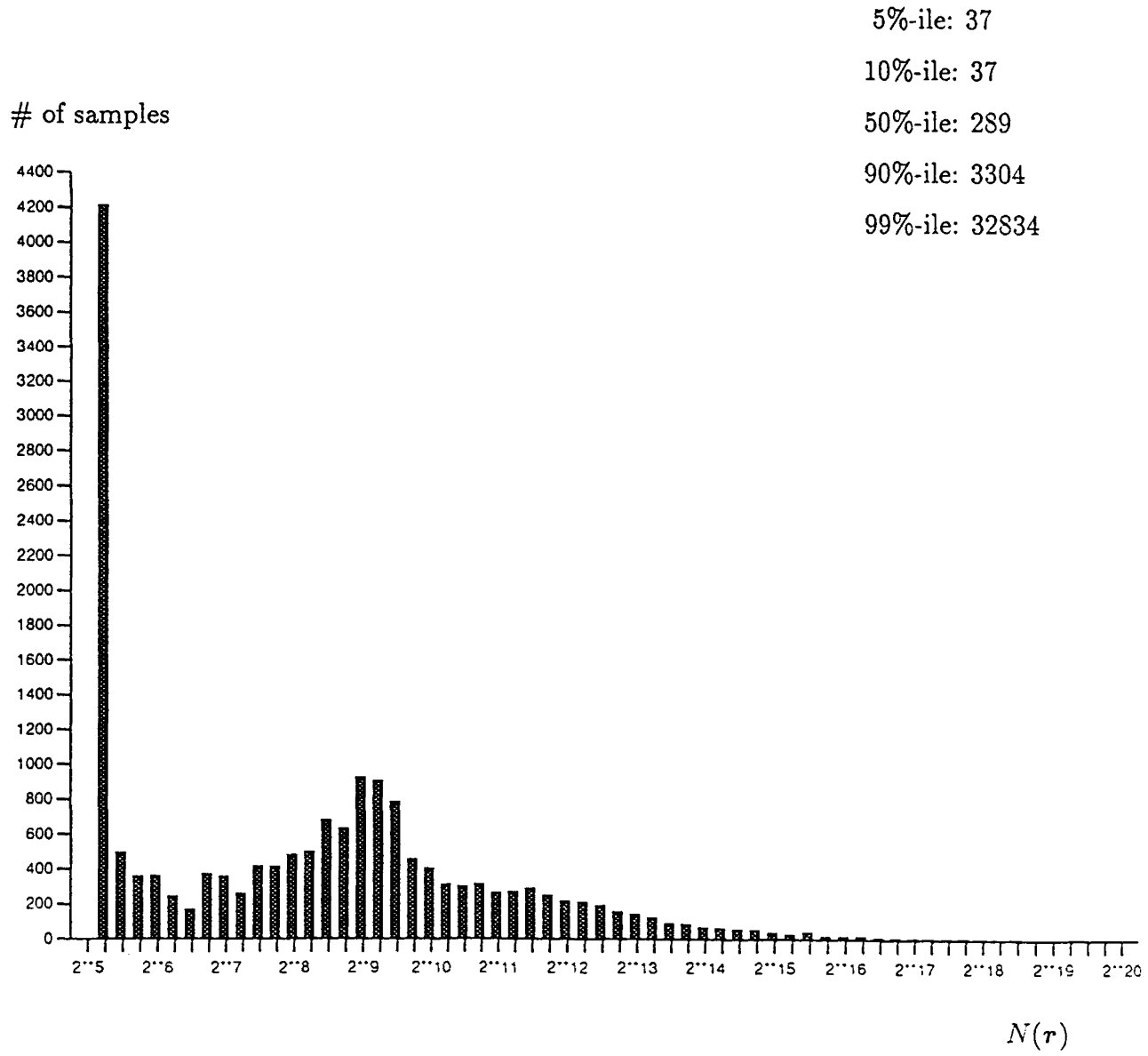


Fig. 4. Histogram representing $N(\boldsymbol{r})$ for 17,000 received vectors obtained by transmitting the (72,36) code over **AWGN** channel at $\gamma_b = 3\ dB$.

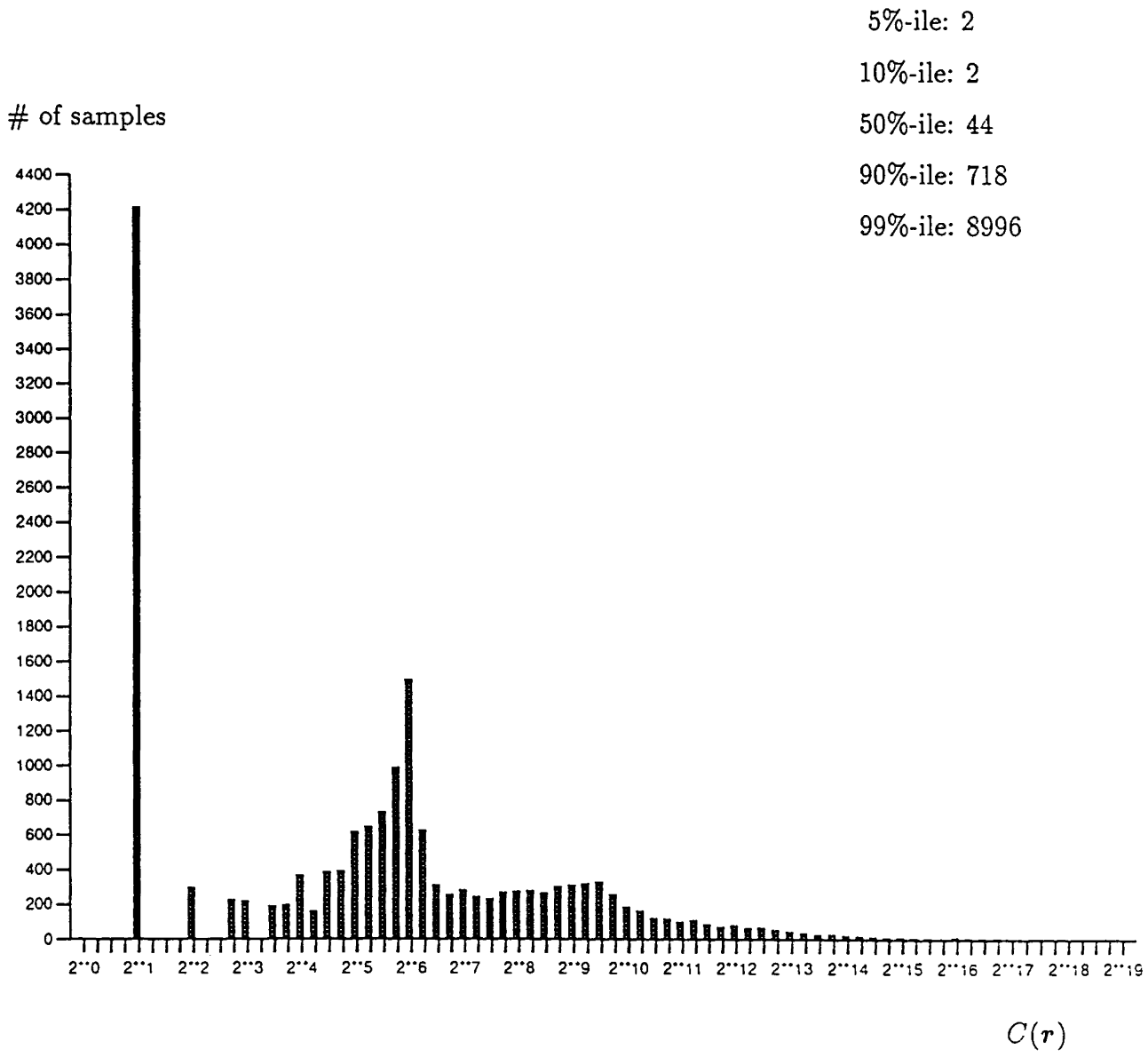


Fig. 5. Histogram representing $C(\boldsymbol{r})$ for 17,000 received vectors obtained by transmitting the (72,36) code over AWGN channel at $\gamma_b = 3\ dB$.

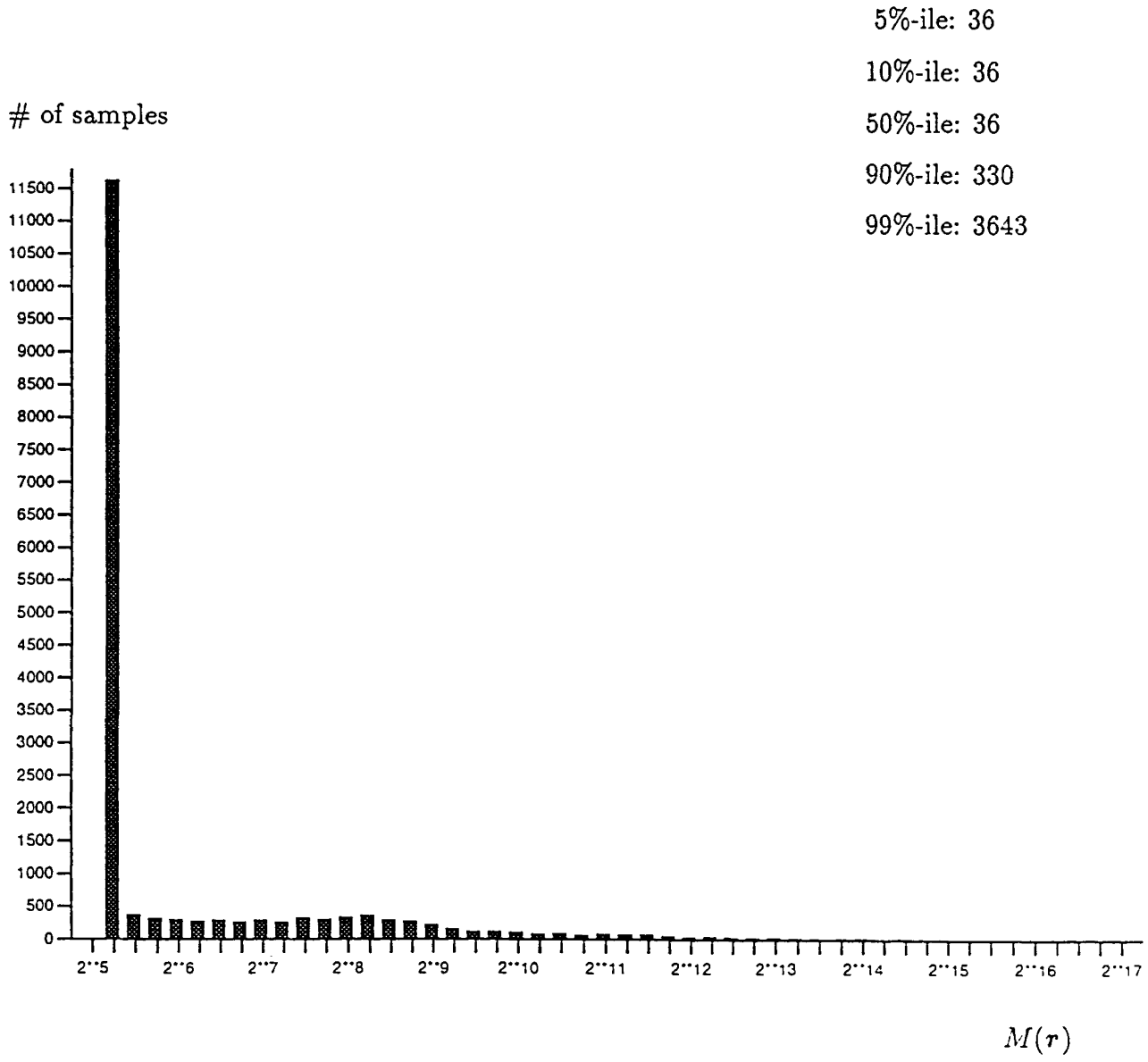


Fig. 6. Histogram representing $M(\boldsymbol{r})$ for 17,000 received vectors obtained by transmitting the (72,36) code over AWGN channel at $\gamma_b = 3\ dB$.

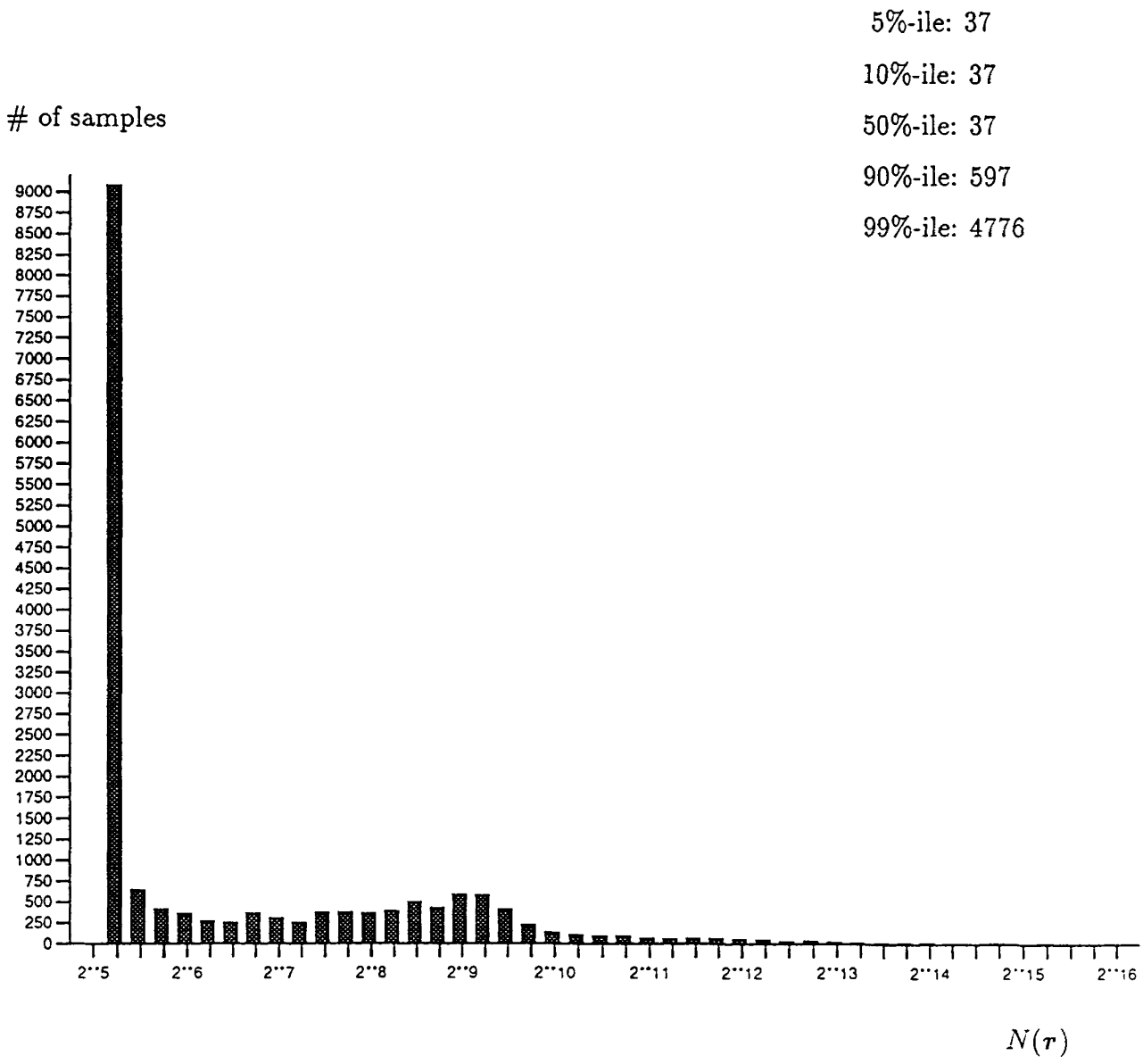


Fig. 7. Histogram representing $N(\boldsymbol{r})$ for 17,000 received vectors obtained by transmitting the (72,36) code over AWGN channel at $\gamma_b = 4\ dB$.

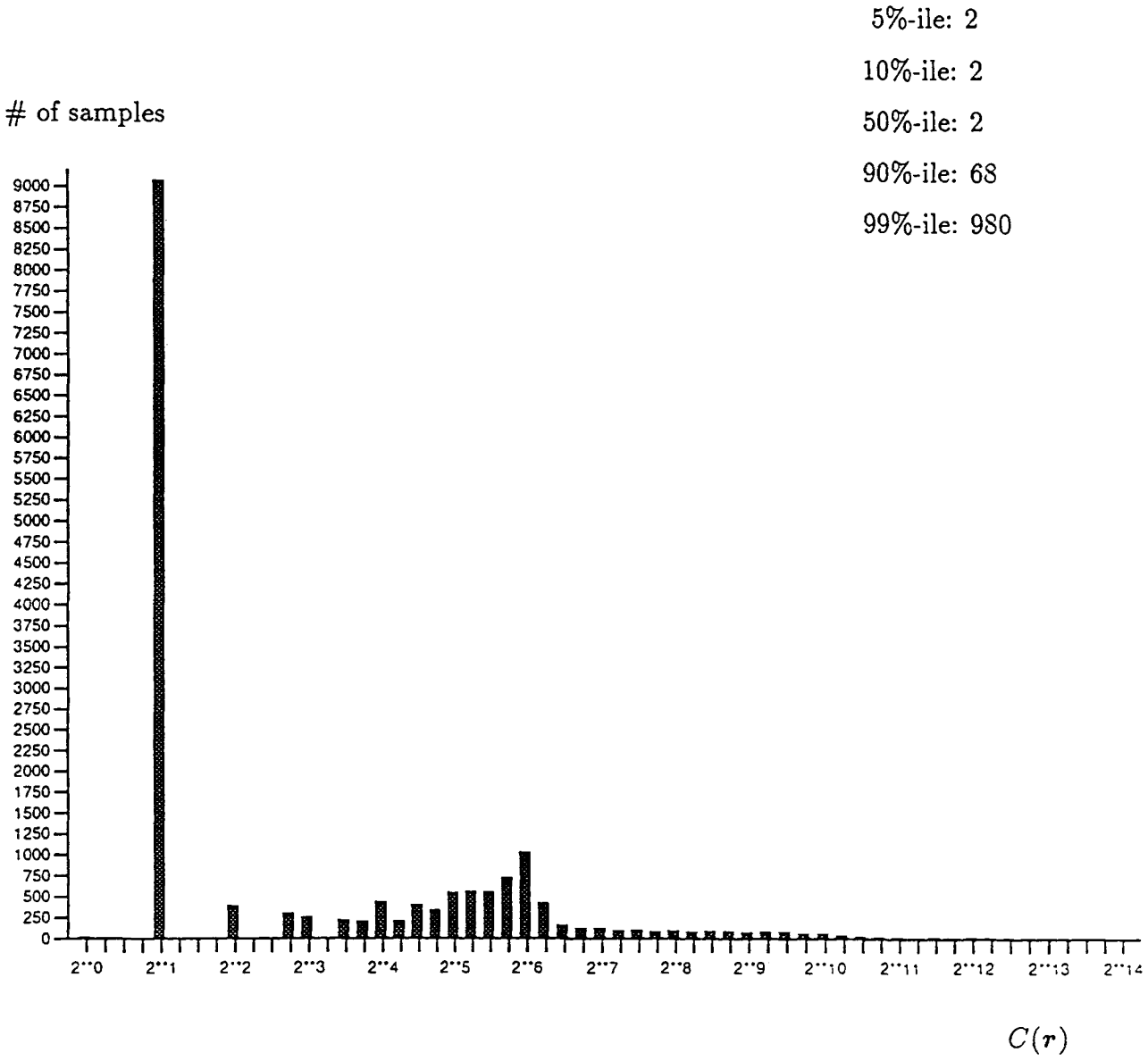


Fig. 8. Histogram representing $C(\boldsymbol{r})$ for 17,000 received vectors obtained by transmitting the (72,36) code over AWGN channel at $\gamma_b = 4\ dB$.

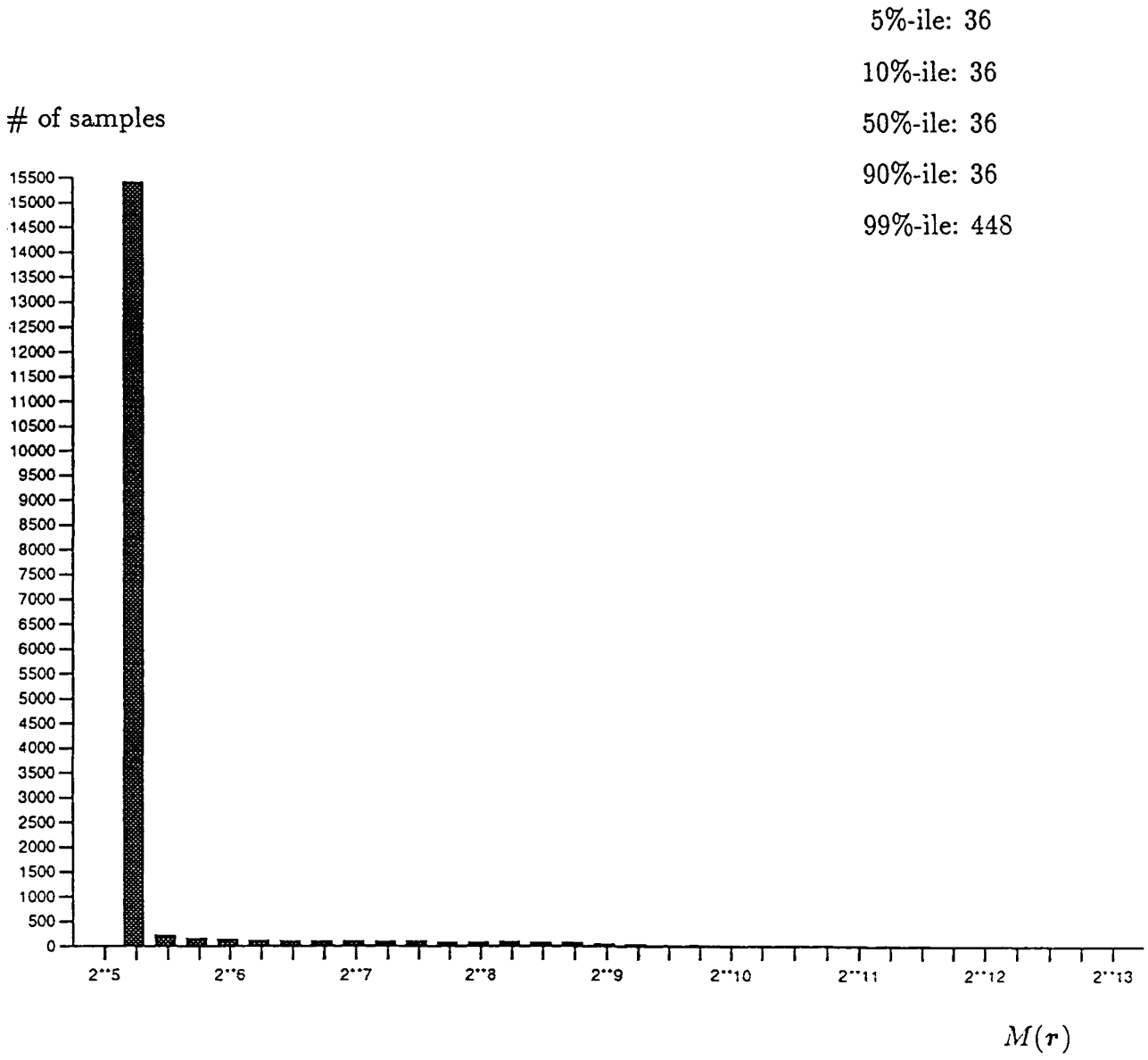


Fig. 9. Histogram representing $M(\boldsymbol{r})$ for 17,000 received vectors obtained by transmitting the (72,36) code over AWGN channel at $\gamma_b = 4\ dB$.

# References


[1] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Trans. on Information Theory*, pp. 284–287, March 1974.

[2] L. D. Baumert and L. R. Welch, "Minimum-Weight Codewords in the (128,64) BCH Code," DSN Progress Report 42–42, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, September and October 1977.

[3] L. D. Baumert and R. J. McEliece, "Soft Decision Decoding of Block Codes," DSN Progress Report 42–47, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, July and August 1978.

[4] E. R. Berlekamp, *Algebraic Coding Theory.* New York, NY: McGraw-Hill Book Co., 1968.

[5] R. E. Blahut, *Theory and Practice of Error Control Codes.* Reading, MA: Addison-Wesley Publishing Co., 1983.

[6] R.W. D. Booth, M. A. Herro, and G. Solomon, "Convolutional Coding Techniques for Certain Quadratic Residue Codes," in *Proc. 1975 Int. Telemetering Conf.*, pp. 168–177, 1975.

[7] G. C. Clark, Jr., and J. B. Cain, *Error-Correction Coding for Digital Communications.* New York, NY: Plenum Press, 1981.

[8] J. H. Conway and N. J. A. Sloane, "Soft Decoding Techniques for Codes and Lattices, Including the Golay Code and the Leech Lattice," *IEEE Tran. on Information Theory*, pp. 41–50, January 1986.

[9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms.* Cambridge, MA: The MIT Press, 1991.

[10] B. G. Dorsch, "A Decoding Algorithm for Binary Block Codes and $J$-ary Output Channels," *IEEE Trans. Information Theory*, pp. 391–394, May 1974.

[11] G. D. Forney, Jr., *Concatenated Codes.* Cambridge, MA: The M.I.T. Press, 1966.

[12] T.-Y. Hwang, "Decoding Linear Block Codes for Minimizing Word Error Rate," *IEEE Trans. on Information Theory*, pp. 733–737, November 1979.

[13] T.-Y. Hwang, "Efficient Optimal Decoding of Linear Block Codes," *IEEE Trans. on Information Theory*, pp. 603–606, September 1980.

[14] L. B. Levitin, M. Naidjate, and C. R. P. Hartmann, "Generalized Identity-Guards Algorithm for Minimum Distance Decoding of Group Codes in Metric Space," presented at the 1990 IEEE International Symposium on Information Theory, San Diego, CA, January 1990.

[15] J. L. Massey, "Variable-Length Codes and the Fano Metric," *IEEE Trans. on Information Theory*, pp. 196–198, January 1972.

[16] M. Naidjate, "Generalized Minimum Distance Decoding Algorithms for Group Codes in Metric Spaces," Ph.D. dissertation, Boston University, Boston, MA, 1991.

[17] N. J. Nilsson, *Principle of Artificial Intelligence.* Palo Alto, CA: Tioga Publishing Co., 1980.

[18] G. Solomon and H. C. A. van Tilborg, "A Connection Between Block and Convolutional Codes," *SIAM J. Appl. Math.*, pp. 358–369, 1979.

[19] A. M. Tenenbaum and M. J. Augenstein, *Data Structures Using Pascal.* Second edition. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1986.

[20] A. J. Viterbi, "Error Bound for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Trans. on Information Theory*, pp. 260–269, April 1967.

[21] J. K. Wolf, "Efficient Maximum Likelihood Decoding of Linear Block Codes Using a Trellis," *IEEE Trans. on Information Theory*, pp. 76–80, January 1978.