7-1996

Efficient Heuristic Search Algorithms for Soft-Decision Decoding of Linear Block Codes

Ching-Cheng Shih
Syracuse University

C. R. Wulff

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

Chilukuri K. Mohan
Syracuse University, ckmohan@syr.edu

SU-CIS-96-3

Efficient Heuristic Search Algorithms for Soft-Decision Decoding of Linear Block Codes

C.-C. Shih, C. R. Wulff,
C. R. P. Hartmann, and C. K. Mohan

July 1996

School of Computer and Information Science
Syracuse University
Suite 2-120, Center for Science and Technology
Syracuse, New York 13244-4100

Efficient Heuristic Search Algorithms for Soft-Decision Decoding of Linear Block Codes[1]

Ching-Cheng Shih

Christopher R. Wulff

Carlos R. P. Hartmann

Chilukuri K. Mohan

School of Computer and Information Science

Syracuse University, Syracuse, NY 13244-4100, USA

Email: ccshih/crwulff/hartmann/mohan@top.cis.syr.edu

Telephone: (315) 443-2368

Fax: (315) 443-1122

[1]This work was partially supported by the National Science Foundation under Grant NCR-9205422.

Abstract

This paper deals with *maximum-likelihood soft-decision* decoding as well as *suboptimal soft-decision* decoding of linear block codes. In this paper we present a novel and efficient hybrid decoding algorithm for (n, k) linear block codes. This algorithm consists of three new decoding algorithms: MA^*, H^*, and Directed Search. It hybridizes these three algorithms to take advantage of their strengths and make the decoding more efficient. The first algorithm, MA^*, is a modified Algorithm A^* that conducts a heuristic search through a code tree of the transmitted code when the decoding problem is transformed into a problem of graph-search through a code tree. MA^* takes into consideration more properties of the code and is considerably more efficient than the original A^* algorithm presented by Han, Hartmann, and Chen. The second algorithm, H^*, is a new decoding algorithm that determines the value of every component of a minimum-cost codeword by estimating the cost of the minimum-cost codeword, which has a fixed value at one of the k most reliable, linearly independent bit positions when the decoding problem is transformed into a minimum-cost problem among all codewords of the transmitted code. The suboptimal version of this algorithm can be incorporated with other decoding algorithms to reduce the search space during the decoding process. The third algorithm, Directed Search, is a novel heuristic approach designed to enhance the performance of soft-decision decoding by searching in continuous space. This approach explores the search space between a given vector and the received vector and finds the closest codeword to the received vector in the space explored. Simulation results for this hybrid algorithm are presented for the $(128, 64)$, the $(256, 131)$, and the $(256, 139)$ binary-extended BCH codes. This hybrid algorithm can efficiently decode the $(128, 64)$ code for any signal-to-noise ratio and has near-optimal to optimal performance. Previously, no practical decoder could have decoded this code with such a performance for all ranges of signal-to-noise ratio.

Index terms: block codes, soft-decision, decoding, code tree, graph-search, BCH code

Contents

1 Introduction

The use of block codes is a well-known error-control technique for reliable transmission of digital information over noisy communication channels. Linear block codes with good coding gains have been known for many years; however, these block codes have not been used in practice due to the unavailability of an efficient soft-decision decoding algorithm.

This paper deals with *maximum-likelihood soft-decision* (MLSD) decoding as well as *suboptimal soft-decision* (SOSD) decoding of linear block codes. By *maximum-likelihood decoding* (MLD), we mean the minimization of the probability of decoding to an incorrect codeword when all codewords have equal probability of being transmitted. By *soft-decision* we mean the use of real numbers (e.g., the analog output of filters matched to signals) associated with every component of the codeword in the decoding procedure. Soft-decision decoding can provide about 2 dB of additional coding gain when compared with hard-decision decoding [4].

Several researchers [1, 34, 30] have presented techniques for decoding linear block codes that convert the decoding problem into a problem of graph-search on a trellis derived from the parity-check matrix of the code. Thus the MLD rule can be implemented by applying the Viterbi Algorithm [33] to this trellis. In practice, however, this breadth-first search scheme can be applied only to codes with small redundancy or to codes with a small number of codewords [25].

The decoding problem has been converted into a problem of search through a code tree for an equivalent code of the transmitted code [14]. In [28, 29, 7, 20, 13, 17, 11, 14] several decoding algorithms for linear block codes were proposed. These algorithms are efficient for codes of short-to-moderate lengths, but become impractical for codes of long lengths transmitted over channels with low SNR; so we still lack an efficient decoding algorithm for long linear block codes. The use of long block codes in the design of communication channels is important because with noise averaging we expect codes of greater lengths to be more "effective" than codes of shorter lengths [4, Sec. 1.1].

In this paper we present a novel and efficient hybrid decoding algorithm for linear block

codes. This algorithm consists of three new decoding algorithms: MA^*, H^*, and Directed Search. It hybridizes MA^*, H^*, and Directed Search to take advantage of their strengths and make decoding more efficient. The first algorithm, MA^*, is a modified Algorithm A^* which searches through a code tree of the transmitted code when the decoding problem is converted into a problem of graph-search through a code tree. MA^* takes into consideration more properties of the code and is considerably more efficient than the original A^* algorithm [14]. The second algorithm, H^*, is a new decoding algorithm which determines the value of every component of a minimum-cost codeword by estimating the cost of the minimum-cost codeword, which has a fixed value at one of the first k most reliable, linearly independent bit positions when the decoding algorithm is transformed into a minimum-cost problem among all codewords of the transmitted code. The suboptimal version of this algorithm can be incorporated with other decoding algorithms to reduce the search space during the decoding process. The third algorithm, Directed Search, is a novel iterative procedure designed to enhance the performance of soft-decision decoding by searching in continuous space. This approach explores the search space between a given vector and the received vector and finds the closest codeword to the received vector in the space explored. In each iteration of the procedure, real vectors successively closer to the received vector are examined, simultaneously constructing the corresponding codewords at local minima of the cost function, and storing the most recent and closest codewords to the received vector examined so far. When the decoding problem is converted into a continuous optimization problem, Directed Search overcomes the main problems of simple hill-climbing and gradient-descent algorithms [6], as well as those of simulated annealing [21] and Tabu Search [10], and directs the search toward the received vector while escaping from a local minimum. This approach takes advantage of the properties of the code and successfully works on both k-dimensional and n-dimensional spaces to make the search more efficient.

We emphasize that the decoding algorithms presented in this paper are suitable for decoding any linear block codes. We present simulation results for the codes with length equal to 128 or 256. For linear block codes with length smaller than 128 and dimension smaller than 64, such as the $(104, 52)$ binary-extended quadratic residue code, MA^* achieves

near-optimal to optimal performance for any signal-to-noise ratio value as we have presented in our previous work [29].

In Section 2 we review maximum-likelihood decoding of linear block codes. The transformations of the decoding problem into a problem of graph-search, as well as a continuous optimization problem, are also presented in this section. MA^* is described in Section 3, H^* is described in Section 4, and Directed Search is presented in Section 5. The hybrid algorithm is presented in Section 6. Simulation results of this hybrid algorithm for the $(128, 64)$, the $(256, 131)$, and the $(256, 139)$ binary extended BCH codes are given in Section 7. Concluding remarks are presented in Section 8.

2 Preliminaries

Let $\boldsymbol{C}$ be a binary (n, k) linear code with generator matrix $\boldsymbol{G}$, and let $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$ 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 assume that $\boldsymbol{Pr}(r_j|c_i) > 0$ for $r_j \in \boldsymbol{B}$ and $c_i \in \boldsymbol{GF}(2)$. The MLSD decoding rule for a time-discrete, memoryless channel can be formulated as [19]: given a received vector $\boldsymbol{r} = (r_0, r_1, \ldots, r_{n-1})$, find a codeword $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1}) \in \boldsymbol{C}$ that minimizes $\sum_{j=0}^{n-1} (\phi_j - (-1)^{c_j})^2$, where $\phi_j = \ln \frac{\boldsymbol{Pr}(r_j|0)}{\boldsymbol{Pr}(r_j|1)}$. We may therefore consider that the "received vector" is $\boldsymbol{\phi} = (\phi_0, \phi_1, \ldots, \phi_{n-1})$. Furthermore, the MLSD decoding problem is equivalent to the minimum distance problem if we define the distance between $\boldsymbol{\phi}$ and a codeword $\boldsymbol{c} \in \boldsymbol{C}$ as the Euclidean distance between $\boldsymbol{\phi}$ and the binary n-tuple vector $((-1)^{c_0}, (-1)^{c_1}, \ldots, (-1)^{c_{n-1}}) \in \{-1, 1\}^n$. We say that a codeword $\boldsymbol{c} \in \boldsymbol{C}$ is "closer" to $\boldsymbol{\phi}$ than another codeword $\boldsymbol{c'} = (c'_0, c'_1, \ldots, c'_{n-1}) \in \boldsymbol{C}$ if and only if $\sum_{j=0}^{n-1} (\phi_j - (-1)^{c_j})^2 \leq \sum_{j=0}^{n-1} \left(\phi_j - (-1)^{c'_j}\right)^2$.

Simulation results [6, 14] showed that it is very important that we select the "best" set of linearly independent codeword variables, and this best set corresponds to the set of the k most reliable, linearly independent positions in the received vector $\boldsymbol{\phi}$, where ϕ_i is said to be more reliable than ϕ_j if and only if $| \phi_i | > | \phi_j |$. It is important to note that each time a new vector is received, the k most reliable, linearly independent positions must be determined

anew. For simplicity of notation and without loss of generality, we will henceforth take $\phi_0, \phi_1, \ldots, \phi_{k-1}$ to be the set of k most reliable, linearly independent positions with $\mid \phi_j \mid \geq \mid \phi_{j+1} \mid$ for $j = 0, 1, \ldots k-2$. Furthermore, we will assume that the first k columns of the generator matrix $\boldsymbol{G}$ form a $k \times k$ identity matrix, i.e., $\boldsymbol{G} = [\boldsymbol{I_k} \boldsymbol{A}]$, where $\boldsymbol{I_k}$ is the $k \times k$ identity matrix.

2.1 Decoding as a Discrete Optimization Problem

When the decoding problem is converted into a problem of graph-search through a code tree for code $\boldsymbol{C}$, we attempt to find a minimum-cost path from the start node to a goal node in the code tree. A code tree is a way to represent every codeword of an (n, k) linear code $\boldsymbol{C}$ as a path through a code tree containing $n+1$ levels. The root is called the *start node*, which is at level -1. There are two branches, labeled 0 and 1, respectively, that leave each node at the first k levels. After the first k levels, there is only one branch leaving each node. The 2^k leaves are called *goal nodes*, which are at level $n-1$. Let $c_0, c_1, \ldots, c_{k-1}$ be the sequence of labels encountered when traversing a path from the start node to a node m at level $k-1$. Then $c_k, c_{k+1}, \ldots, c_{n-1}$, the sequence of labels encountered when traversing a path from node m to a goal node, can be obtained as follows: $(c_0, c_1, \ldots, c_{k-1}, c_k, \ldots, c_{n-1}) = (c_0, c_1, \ldots, c_{k-1})\ \boldsymbol{G}$.

We now specify the arc cost in the code tree of $\boldsymbol{C}$. The arc from a node at level $t-1$ to its immediate successor in a path is assigned cost $(\phi_t - (-1)^{c_t})^2$, where c_t is the label of the arc. This arc is called the t^{th} arc of the path. Thus, the decoding problem is equivalent to finding an "optimal" path from the start node to a goal node in the code tree which corresponds to a codeword $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$ such that $\sum_{j=0}^{n-1} (\phi_j - (-1)^{c_j})^2$ is minimum among all paths from the start node to the goal nodes.

We note that an optimal path in the code tree corresponds to a codeword that minimizes $\sum_{j=0}^{n-1} (\phi_j - (-1)^{c_j})^2$, where the label for the j^{th} arc of a path corresponds to the value of the j^{th} component of a codeword. If we define the cost for the j^{th} component of a codeword as the cost of the j^{th} arc of the corresponding path in the code tree and the cost for a codeword as the sum of the cost for all components of this codeword, the decoding problem becomes

equivalent to the minimum-cost problem among all codewords in $\boldsymbol{C}$, i.e., finding a codeword $\boldsymbol{c} \in \boldsymbol{C}$ that minimizes the cost $\sum_{j=0}^{n-1}(\phi_j - (-1)^{c_j})^2$.

2.2 Decoding as a Continuous Optimization Problem

We would like to transform the decoding problem into a continuous optimization problem to take advantage of more information provided by the received vector and the inherent properties of the transmitted code that cannot be taken into consideration when the decoding problem is solved as a discrete optimization problem. Furthermore, it is often easier to optimize a function over a continuous domain (using the tools of calculus or linear programming) than over a discrete domain [26, Sec. 4.4], [27].

In order to transform the decoding problem into a continuous optimization problem [6], we apply the mapping $z \longrightarrow (-1)^z$ from $\boldsymbol{GF}(2)$ to the real field R. Then $\boldsymbol{C}$ is mapped to $\boldsymbol{C'}$, where a $\{-1,1\}$-vector $\boldsymbol{c'} = (c'_0, c'_1, \ldots, c'_{n-1})$ is a codeword of $\boldsymbol{C'}$ if and only if $\prod_{j=0}^{n-1}(c'_j)^{h_{ij}} = 1$ for $i = 0, 1, \ldots, n-k-1$, where $\boldsymbol{H} = [h_{ij}]$ is the parity check matrix of $\boldsymbol{C}$. In this formulation, the MLSD decoding rule for a time-discrete, memoryless channel becomes: Given a received vector $\boldsymbol{\phi} = (\phi_0, \phi_1, \ldots, \phi_{n-1})$, find a codeword $\boldsymbol{c'} \in \boldsymbol{C'}$ that minimizes $\sum_{j=0}^{n-1}(\phi_j - c'_j)^2$.

Note that $\sum_{j=0}^{n-1}(\phi_j - c'_j)^2 = \sum_{j=0}^{n-1}\phi_j^2 + n - 2\sum_{j=0}^{n-1}\phi_j c'_j$, since $\sum_{j=0}^{n-1}(c'_j)^2 = n$ for all $\boldsymbol{c'} \in \boldsymbol{C'}$. Furthermore, the dependent codeword variables $c'_k, c'_{k+1}, \ldots, c'_{n-1}$ are related to the linearly independent codeword variables $c'_0, c'_1, \ldots, c'_{k-1}$ by

$$c'_i = \pi_i(c'_0, c'_1, \ldots, c'_{k-1}) = \prod_{j=0}^{k-1}(c'_j)^{p_{ij}}, \text{ for } i = k, k+1, \ldots, n-1, \tag{1}$$

where the $p_{ij} \in \{0,1\}$ are determined from the $n-k$ parity product equations, which correspond to the first k most reliable, linearly independent positions of $\boldsymbol{\phi}$. Thus, for a codeword $\boldsymbol{c'} \in \boldsymbol{C'}$ we may write $\sum_{j=0}^{n-1}\phi_j c'_j = \sum_{j=0}^{k-1}\phi_j c'_j + \sum_{j=k}^{n-1}\phi_j \pi_j(c'_0, c'_1, \ldots, c'_{k-1})$, which is a function of the k most reliable, linearly independent codeword variables $c'_0, c'_1, \ldots, c'_{k-1}$. Then the optimization problem in R^k is: Given a received vector $\boldsymbol{\phi}$, find a vector $(c'_0, c'_1, \ldots, c'_{k-1}) \in \{-1, +1\}^k$ that minimizes the cost function f_ϕ defined as

$$f_\phi(c'_0, c'_1, \ldots, c'_{k-1}) = \sum_{j=0}^{n-1}\phi_j^2 + n - 2\sum_{j=0}^{k-1}\phi_j c'_j - 2\sum_{j=k}^{n-1}\phi_j \pi_j(c'_0, c'_1, \ldots, c'_{k-1}).$$

When we restrict attention to the discrete space, $\{-1,+1\}^k$, over all vertices of the solid k-cube of length two centered at the origin, we have established that if $(c'_0, c'_1, \ldots, c'_{k-1})$ minimizes f_ϕ, then the codeword generated by $c'_0, c'_1, \ldots, c'_{k-1}$ according to Equation (1) is the closest codeword to the received vector ϕ. This is still a discrete optimization problem, but if the cost function f_ϕ is allowed to range over all points in the solid k-cube defined by $\mid x_j \mid \leq 1$, $j = 0, 1, \ldots, k-1$, it still takes its minimum at a vertex. This follows from the fact that $f_\phi(x_0, x_1, \ldots, x_{k-1})$ is linear in each x_i. Therefore, every local minimum of f_ϕ corresponds to a codeword in C', and the value of f_ϕ at every local minimum is equal to the square of the Euclidean distance between the received vector and the codeword corresponding to the local minimum. In this way we arrived at the MLSD decoding rule in R^k: Given a received vector ϕ, find a real vector $(x_0, x_1, \ldots, x_{k-1})$ in the solid k-cube defined by $\mid x_j \mid \leq 1$, $j = 0, 1, \ldots, k-1$ that minimizes the cost function f_ϕ. We note that a real vector $(x_0, x_1, \ldots, x_{k-1})$ has a one-to-one corresponding real vector $(x_0, x_1, \ldots, x_{k-1}, \ldots, x_{n-1}) \in R^n$ such that the last $n-k$ components are obtained from equation (1).

First we describe the algorithms which compose the hybrid decoding algorithm.

3 Modified Algorithm A^* (MA^*)

In this section we describe MA^*, a new decoding algorithm that searches through a code tree of the transmitted code when the decoding problem is converted into a problem of graph-search through a code tree. MA^* takes into consideration more properties of the code and is considerably more efficient than the original A^* algorithm [14].

When the decoding problem is converted into a problem of graph-search through a code tree, we are interested in finding an optimal path from the start node to a goal node in this code tree. MA^*, which uses a priority-first search strategy, is employed to search through this code tree. For a fixed positive integer q, two functions, f_q and low_q, are defined for every node in the code tree to take advantage of the information provided by the received vector and the inherent properties of the transmitted code. In MA^*, search is guided by the evaluation function f_q. The algorithm maintains a list $\mathcal{L}$ of nodes of the code tree that

are candidates to be expanded. The algorithm selects for expansion the node in $\mathcal{L}$ with minimum values of function f_q. If it selects a goal node for expansion, it has found an "optimal" path from the start node to a goal node whose labels correspond to a codeword that minimizes the error probability when we assume all codewords have equal probability of being transmitted. Function low_q is a new lower bound on the cost of an optimal path that goes through a node in the code tree. The value of low_q is always greater than or equal to the value of f_q, i.e., for every node m, $low_q(m) \geq f_q(m)$. This algorithm also keeps an upper bound (UB) on the cost of an optimal path. If the value of low_q for a node is greater than or equal to the UB, no further search through this node is necessary and this node can be discarded.

For every node m in the code tree and a fixed positive integer q, functions f_q and low_q are defined as

$$f_q(m) = g(m) + h_q(m)$$

and

$$low_q(m) = g(m) + h'_q(m),$$

where $g(m)$ is the actual cost of the path from the start node s to node m, $h_q(m)$ estimates the cost of the minimum-cost path from node m to a goal node, and $h'_q(m)$ estimates the cost of the minimum-cost path from node m to a goal node that takes into consideration the parity check property of the code.

For a positive integer q, we now give a definition for this new heuristic function h_q, which takes into consideration q binary n-tuple vectors

1. that satisfy the Hamming weight constraints,

2. whose corresponding paths from level 0 to level $k-1$ in the code tree go through node m, and

3. that are closer to the received vector ϕ than any other binary n-tuple vectors.

Let $HW = \{w_i | 0 \leq i \leq J\}$ be the set of all $J+1$ distinct Hamming weights that codewords of $\boldsymbol{C}$ may have. Furthermore, assume $w_0 < w_1 < \cdots < w_J$. Our new heuristic function h_q is defined to take into consideration the linear property of $\boldsymbol{C}$ and the fact that the Hamming distance between any two codewords of $\boldsymbol{C}$ must belong to HW. Heuristic function h_q is defined with respect to a codeword $\boldsymbol{c}_{seed} \in \boldsymbol{C}$ which is called the seed of the decoding algorithm.

1. For nodes at level s, $0 \leq s < k$:

   Let m be a node at level s and let $\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_s$ be the labels of the path $\boldsymbol{P}_m$ from the start node to node m. Let $T(m)$ be the set of all binary n-tuple vectors $\boldsymbol{v}$ whose first $s+1$ entries are the labels of $\boldsymbol{P}_m$ and $d_H(\boldsymbol{v}, \boldsymbol{c}_{seed}) \in HW$, where $d_H(\boldsymbol{x}, \boldsymbol{y})$ is the Hamming distance between $\boldsymbol{x}$ and $\boldsymbol{y}$. That is,

   $$T(m) = \{\boldsymbol{v} | \boldsymbol{v} = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_s, v_{s+1}, \ldots, v_{n-1}) \text{ and } d_H(\boldsymbol{v}, \boldsymbol{c}_{seed}) \in HW\}.$$

   We will construct a sequence of q finite nonempty subsets of $T(m)$, $T_1(m) \supseteq T_2(m) \supseteq \ldots \supseteq T_q(m)$, and for every $T_i(m)$ we define $\boldsymbol{v}_i = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_s, v_{i(s+1)}, \ldots, v_{i(n-1)}) \in T_i(m)$ as the vector that satisfies

   $$\sum_{j=s+1}^{n-1} (\phi_j - (-1)^{v_{ij}})^2 = \min_{\boldsymbol{v} \in T_i(m)} \left\{ \sum_{j=s+1}^{n-1} (\phi_j - (-1)^{v_j})^2 \right\}.$$

   For $1 \leq i \leq q$, $T_i(m)$ is now recursively defined as

   (a) $T_1(m) = T(m)$;

   (b) for $1 \leq i < q$,

   $$T_{i+1}(m) = \begin{cases} T_i(m) & \text{if } \boldsymbol{v}_i \text{ is a codeword;} \\ T_i(m) - \{\boldsymbol{v}_i\} & \text{otherwise.} \end{cases}$$

   The heuristic function h_q is defined as

   $$h_q(m) = \sum_{j=s+1}^{n-1} (\phi_j - (-1)^{v_{qj}})^2, \text{ where } \boldsymbol{v}_q = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_s, v_{q(s+1)}, \ldots, v_{q(n-1)}).$$

   Note that $T_i(m) \neq \emptyset$, since $\boldsymbol{u}\,\boldsymbol{G}$ is a codeword in $T(m)$, where $\boldsymbol{u} = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_s, 0, \ldots, 0)$ is a binary k-tuple vector.

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

   Let m be a node at level s. We define $h_q(m)$ as the actual cost of the unique path from node m to a goal node, i.e., $h_q(m) = \sum_{i=s+1}^{n-1} (\phi_i - (-1)^{v_{qi}})^2$, where $\boldsymbol{v}_q = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_s, v_{q(s+1)}, \ldots, v_{q(n-1)}) = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_{k-1})\ \boldsymbol{G}$, and $v_{q(s+1)}, v_{q(s+2)}, \ldots, v_{q(n-1)}$ are the sequence of labels of the unique path $\boldsymbol{P}_m$ from node m to a goal node.

Function h'_q is defined analogously as h_q, except that it takes into consideration the fact that components in the redundancy part of a codeword can be determined by parity checks. Let $\boldsymbol{H} = [h_{ij}]$ be the parity check matrix of code $\boldsymbol{C}$ in canonical form, i.e., the last $n-k$ columns of $\boldsymbol{H}$ form an identity matrix. Let $S_H = \{\boldsymbol{h}_0, \boldsymbol{h}_1, \ldots, \boldsymbol{h}_{n-k-1} \mid \boldsymbol{h}_i = (h_{i0}, h_{i1}, \ldots, h_{i(n-1)})$ is the i^{th} row of $\boldsymbol{H}\}$. For a node m at level s, $0 \leq s < k$, such that $\overline{v'}_0, \overline{v'}_1, \ldots, \overline{v'}_s$ is the sequence of labels of the path $\boldsymbol{P}_m$ from the start node to node m, we define $S_H(m) = \{\boldsymbol{h}_i \mid \boldsymbol{h}_i \in S_H,\ h_{ij} = 0$ for $j = s+1, s+2, \ldots, k-1\}$. According to the linear property of the code, if $S_H(m) \neq \emptyset$, then for any codeword $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1}) \in \boldsymbol{C}$ such that $c_0 = \overline{v'}_0, c_1 = \overline{v'}_1, \ldots, c_s = \overline{v'}_s$, we have $c_{k+i} = \overline{v'}_0 h_{i0} \oplus \overline{v'}_1 h_{i1} \oplus \ldots \oplus \overline{v'}_s h_{is}$ for all $\boldsymbol{h}_i \in S_H(m)$, where $\oplus$ denotes the module 2 addition operator.

Function h'_q can now be defined analogously to h_q except that it is with respect to $T'(m)$ which is the set of all binary n-tuple vectors $\boldsymbol{v}'$

1. whose first $s+1$ entries are $\overline{v'}_0, \overline{v'}_1, \ldots, \overline{v'}_s$,
2. whose $(k+i)^{th}$ entry is $(\overline{v'}_0 h_{i0} \oplus \overline{v'}_1 h_{i1} \oplus \ldots \oplus \overline{v'}_s h_{is})$ for all $\boldsymbol{h}_i \in S_H(m)$,
3. which satisfy the Hamming weight constraints, i.e., $d_H(\boldsymbol{v}', \boldsymbol{c}_{seed}) \in HW$, where $\boldsymbol{c}_{seed}$ is the seed of the decoding algorithm.

That is, $T'(m) = \{\boldsymbol{v}' | \boldsymbol{v}' = (\overline{v'}_0, \overline{v'}_1, \ldots, \overline{v'}_s, v'_{s+1}, \ldots, v'_{n-1}), v'_{k+i} = (\overline{v'}_0 h_{i0} \oplus \overline{v'}_1 h_{i1} \oplus \ldots \oplus \overline{v'}_s h_{is})$ for all $\boldsymbol{h}_i \in S_H(m)$, and $d_H(\boldsymbol{v}, \boldsymbol{c}_{seed}) \in HW\}$. Note that $T'(m) \neq \emptyset$ since the codeword $\boldsymbol{u}\ \boldsymbol{G}$ is also in $T'(m)$, where $\boldsymbol{u} = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_s, 0, \ldots, 0)$ is a binary k-tuple vector. Therefore, if $\boldsymbol{v}'_q = (v'_{q0}, v'_{q1}, \ldots, v'_{q(n-1)})$ is the vector used to calculate $h'_q(m)$, then $v'_{q0} = \overline{v'}_0, v'_{q1} = \overline{v'}_1, \ldots, v'_{qs} = \overline{v'}_s$, and whenever $S_H(m) \neq \emptyset$, then $v'_{q(k+i)} = (\overline{v'}_0 h_{i0} \oplus \overline{v'}_1 h_{i1} \oplus \ldots \oplus \overline{v'}_s h_{is})$ for all $\boldsymbol{h}_i \in S_H(m)$.

Note that if the vector $\boldsymbol{v}'_q$ used to calculate $h'_q(m)$ and the vector $\boldsymbol{v}_q$ used to calculate $h_q(m)$ have different values at bit position $k+i$ for some i, where $\boldsymbol{h}_i \in S_H(m)$ when $S_H(m) \neq \emptyset$, then $h'_q(m) > h_q(m)$; otherwise, $h'_q(m) = h_q(m)$. Furthermore, if node m is a goal node, then $h'_q(m) = h_q(m) = 0$. We now give our MA^* decoding algorithm.

MA^* decoding algorithm

1. Let q be a positive integer and m_{-1} be the start node in the code tree. Generate a nonempty set of codewords $S \subset \boldsymbol{C}$. For each codeword $\boldsymbol{c} \in S$, set $\boldsymbol{c}_{seed} = \boldsymbol{c}$ and calculate $low_q(m_{-1})$. Select the codeword $\boldsymbol{c} \in S$ that maximizes $low_q(m_{-1})$ as the initial seed of the decoding algorithm.

2. For each codeword in S, calculate the cost of the corresponding path from the start node to a goal node in the code tree. Let $\overline{\boldsymbol{c}}$ be the codeword in S that corresponds to the minimum-cost path $\boldsymbol{P}$ among all paths corresponding to codewords in S. Let UB be the cost of $\boldsymbol{P}$. Set list $\boldsymbol{\mathcal{L}} = \emptyset$.

3. Create a subtree that contains only the start node m_{-1}. Calculate $f_q(m_{-1})$. Note that $f_q(m_{-1}) = low_q(m_{-1})$. Put m_{-1} in list $\boldsymbol{\mathcal{L}}$.

4. If list $\boldsymbol{\mathcal{L}}$ is empty, the algorithm terminates and $\boldsymbol{P}$ is an optimal path.

5. Remove the node m from list $\boldsymbol{\mathcal{L}}$, which has minimal value of f_q. If $low_q(m) \geq UB$, go to step 4.

6. Start expanding node m. Assume that node m is at level s in the code tree. Let $\boldsymbol{v}_q = (v_{q0}, v_{q1}, \ldots, v_{q(k-1)}, \ldots, v_{q(n-1)})$ be the vector used to calculate $f_q(m)$. Generate two codewords $\boldsymbol{c}' = (v_{q0}, v_{q1}, \ldots, v_{q(k-1)})\ \boldsymbol{G}$ and $\boldsymbol{c}'' = (v_{q0}, v_{q1}, \ldots, v_{q(k-2)}, v_{q(k-1)} \oplus 1)\ \boldsymbol{G}$. Let UB' be the cost of path $\boldsymbol{P}'$ in the code tree that corresponds to $\boldsymbol{c}'$ and let UB'' be the cost of path $\boldsymbol{P}''$ in the code tree that corresponds to $\boldsymbol{c}''$. If $min\{UB', UB''\} < UB$, set $UB = min\{UB', UB''\}$ and set $\overline{\boldsymbol{c}}$ to be the codeword whose corresponding path has cost $min\{UB', UB''\}$.

7. Set $S = \{\boldsymbol{c}_{seed}, \boldsymbol{c}', \boldsymbol{c}''\}$. Calculate $low_q(m_{-1})$ for every codeword in S. Set $\boldsymbol{c}_{seed}$ to be the one that maximizes $low_q(m_{-1})$ among all codewords in S.

8. Generate the set M which contains all nodes at level i that are not in path $\boldsymbol{P}'$ and are immediate successors of nodes at level $i-1$ in path $\boldsymbol{P}'$ for $i = s+1, s+2, \ldots, k-2$, i.e., $M = \{m_i \mid s < i < k-1$, m_i is a node at level i that is not in path $\boldsymbol{P}'$ and is an immediate successor of the node at level $i-1$ in path $\boldsymbol{P}'$.$\}$.

9. Calculate $low_q(m_i)$ for every node $m_i \in M$. If $low_q(m_i) \geq UB$, discard it from M. If $M = \emptyset$, go to step 4.

10. Calculate $f_q(m_i)$ for every node $m_i \in M$. Insert every node $m_i \in M$ into list $\mathcal{L}$ and reorder[2] nodes so that all nodes in list $\mathcal{L}$ are sorted according to the values of f_q. Set $M = \emptyset$.

11. Go to step 4.

We note that MA^* is a depth-first search algorithm as is the original A^* algorithm. The proof of this property is similar to that given in Appendix E in [14]. Therefore, upper bounds (UBs) on the cost of an optimal path are obtained every time a codeword is generated. The size of list $\mathcal{L}$ can be reduced by discarding nodes whose values of low_q are greater than or equal to the best UB obtained so far by the algorithm. Furthermore, if MA^* doesn't discard any node from an optimal path in step 9, it will find an optimal path before the algorithm terminates, since the code tree is finite and the algorithm keeps the minimum-cost path found so far. The optimality of MA^* is shown in Appendix A. We also note that the values of the low_q of nodes from the start node to a goal node along a path in the code tree are monotonically nondecreasing, i.e., if m_s is an immediate successor of node m_{s-1}, then $low_q(m_{s-1}) \leq low_q(m_s)$. Therefore, if a node is discarded in step 9 of the algorithm, all of its successors can also be discarded. The proof of this property is also given in Appendix A.

[2]We implement list $\mathcal{L}$ using a B-tree, a data structure that efficiently deals with the insertion and deletion of nodes [32].

From the description of MA^*, it is clear that the most important factor in the efficiency of MA^* is the selection of the evaluation function f_q and function low_q. These two functions are used to reduce drastically the search space and to make the decoding efforts of this decoding algorithm adaptable to the noise level. An algorithm to calculate h_q with time complexity $O(qn + q\log(q))$ is given in Appendix B. We note that the evaluation function used in the original Algorithm A^* [14] is a special case of function f_q where q is equal to 1.

If no restriction is placed on the size of list $\mathcal{L}$, then the decoding algorithm based on MA^* is an MLSD decoding algorithm even if in the computation of f_q and low_q the algorithm considers the Hamming weights in a superset of HW. Simulation results have shown that this algorithm drastically reduces the search space; however, it becomes impractical for long linear block codes transmitted over channels with low SNR, since the number of nodes needed to be stored in list $\mathcal{L}$ is still too large. Simulation results have shown that the number of nodes needed to be stored in list $\mathcal{L}$ before an optimal path is found is considerably smaller than the total number of nodes stored before the algorithm terminates. Therefore, as proposed in [16, 13, 11, 15], we may limit the search by setting an upper bound on the size of list $\mathcal{L}$ with small degradations in the performance of the algorithm. In our SOSD decoding algorithm we limit the size of list $\mathcal{L}$, M_B, according to the following criterion.

If a node m needs to be stored in list $\mathcal{L}$ when the size of list $\mathcal{L}$ has reached a given upper bound M_B, then we discard the node with larger f_q value between node m and the node with the maximum value of function f_q in list $\mathcal{L}$.

4 New Decoding Algorithm — H^*

In this section we describe H^*, a new decoding algorithm that determines the value of every component of a minimum-cost codeword by estimating the cost of a minimum-cost codeword with a fixed value at one of the first k most reliable, linearly independent bit positions.

In H^*, an upper bound UB on the cost of a minimum-cost codeword is used to determine the value of every component in the first k-bit positions of this codeword. This algorithm

constructs a lower bound on the cost of codewords whose i^{th} component, $0 \leq i < k$, has a fixed value of δ, $\delta \in \{0,1\}$. If this lower bound is greater than or equal to UB, we may conclude that the value of the i^{th} component of a minimum-cost codeword is $\delta \oplus 1$. This process repeats until the values of the first k components of a minimum-cost codeword have been determined.

Assume that $\boldsymbol{c}^* = (c_0^*, c_1^*, \ldots, c_{n-1}^*)$ is a minimum-cost codeword. Let $\overline{\boldsymbol{c}} = (\overline{c}_0, \overline{c}_1, \ldots, \overline{c}_{n-1})$ be the closest codeword to the received vector found so far by the algorithm. Let UB be the cost of $\overline{\boldsymbol{c}}$. For simplicity of symbols, let $\overline{I} \subset \{0,1,\ldots,n-1\}$ be the set of indices t such that the value of the t^{th} component of a minimum-cost codeword has been determined, i.e., c_t^* is already determined. Let $I = \{0,1,\ldots,k-1\} - \overline{I}$ be the set of indices i, $0 \leq i < k$, in which c_i^* is still undetermined. For every index $i \in I$, function low_j' is defined as

$$low_j'(i, \overline{c}_i) = (\phi_i - (-1)^{\overline{c}_i \oplus 1})^2 + g'(\overline{I}) + h_j''(i),$$

where j is a positive integer, $g'(\overline{I}) = \sum_{j \in \overline{I}} (\phi_j - (-1)^{c_j^*})^2$, and $h_j''(i)$ estimates the sum of the cost for all other undetermined components of $\boldsymbol{c}^*$. Let $\boldsymbol{c}_{seed} \in \boldsymbol{C}$ be the seed of the decoding algorithm. For a positive integer j and every index $i \in I$, function h_j'' is defined analogously to the heuristic function h_j' described in Section 3, except that it is with respect to $T''(i)$, the set of all binary n-tuple vectors $\boldsymbol{v}''$

1. whose i^{th} component is $\overline{c}_i \oplus 1$ for some $i \in I$,
2. whose t^{th} component is c_t^* for every $t \in \overline{I}$,
3. that satisfy the Hamming weight constraints.

That is, $T''(i) = \{\boldsymbol{v}'' \mid \boldsymbol{v}'' = (v_0'', v_1'', \ldots, v_{n-1}''),\ v_i'' = \overline{c}_i \oplus 1,\ v_t'' = c_t^*$ for all $t \in \overline{I}$, and $d_H(\boldsymbol{v}'', \boldsymbol{c}_{seed}) \in HW\}$. Note that $T''(i) \neq \emptyset$, since $\boldsymbol{u}'\,\boldsymbol{G}$ is a codeword in $T''(i)$ where $\boldsymbol{u}' = (u_0', u_1', \ldots, u_{k-1}')$ is a binary k-tuple vector such that $u_i' = \overline{c}_i \oplus 1$, $u_t' = c_t^*$ for all $t \in \overline{I}$, and $u_t'' = 0$ for all $t \in \{0,1,\ldots,k-1\} - \{i\} - \overline{I}$. We also note that function h_j'' is defined for every index $i \in I$, while function h_j' is defined for every node in the code tree. We now give our new decoding algorithm H^*.

H^* decoding algorithm

1. Let p be a positive integer. Generate a nonempty set of codewords $S \subset \boldsymbol{C}$. For every codeword $\boldsymbol{c}$ *in* S, set $\boldsymbol{c}_{seed} = \boldsymbol{c}$ and calculate $low_p(m_{-1})$. Select the codeword $\boldsymbol{c} \in S$ that maximizes $low_p(m_{-1})$ as the seed of the decoding algorithm.

2. Calculate the cost for every codeword in S. Let $\overline{\boldsymbol{c}}$ be the minimum-cost codeword among all codewords in S. Set UB to be the cost of $\overline{\boldsymbol{c}}$. Set $\overline{I} = \emptyset$.

3. Set $I = \{0, 1, \ldots, k-1\} - \overline{I}$.

4. If $I = \emptyset$, the algorithm terminates and $\overline{\boldsymbol{c}}$ is the minimum-cost codeword; otherwise, select an index $i \in I$.

5. Set $j = 1$.

6. Calculate $low'_j(i, \overline{c}_i)$. If $low'_j(i, \overline{c}_i) \geq UB$, set $c^*_i = \overline{c}_i$ and $\overline{I} = \overline{I} \cup \{i\}$. Go to step 3.

7. If the vector $\boldsymbol{v}''_j$ used to calculate $h''_j(i)$ is not a codeword, then set $j = j + 1$ and go to step 6; otherwise, set $\overline{\boldsymbol{c}} = \boldsymbol{v}''_j$ and $UB = low'_j(i, \overline{c}_i)$.

8. If the minimum-cost codeword whose i^{th} component is $\overline{c}_i$ has been generated by the algorithm, then the algorithm terminates and $\overline{\boldsymbol{c}}$ is the minimum-cost codeword.

9. Let $S = \{\boldsymbol{v}''_j, \overline{\boldsymbol{c}}\}$. For each codeword $\boldsymbol{c} \in S$, set $\boldsymbol{c}_{seed} = \boldsymbol{c}$ and calculate $low_p(m_{-1})$. Select the codeword $\boldsymbol{c} \in S$ that maximizes $low_p(m_{-1})$ as the seed of the decoding algorithm. Go to step 5.

During the decoding process, the search space shrinks by half when the value for one component of the minimum-cost codeword is determined and fixed. The optimality of this algorithm is given in Appendix C. We note that the number of vectors needed to be generated before the algorithm finds a minimum-cost codeword is of order $O(2^n)$. In our suboptimal version of the H^* algorithm, we limit the number of vectors generated in determining the value of a component of a minimum-cost codeword such that the algorithm generates at most

p vectors in determining the value of a fixed component, where p is a fixed positive integer. This suboptimal H^* algorithm can be incorporated with other decoding algorithms to reduce the search space during the decoding process and to make the decoding more efficient.

5 Directed Search (DS)

In this section, we describe a novel iterative procedure designed to enhance the performance of soft-decision decoding by searching in continuous space. This approach explores the search space between a given vector and the received vector and finds the codeword closest to the received vector in the space explored. In each iteration of the procedure, real vectors successively closer to the received vector are examined, simultaneously constructing the corresponding codewords at local minima of the cost function f_ϕ, and storing the most recent and closest codewords to the received vector examined so far.

For the decoding problem that is converted into a continuous optimization problem, a decoding scheme that applies the simple hill-climbing or gradient-descent algorithm to solve the decoding problem was proposed in [6]. The obvious pitfalls of that approach are the large number of local minima of the function being optimized and the fact that the algorithm always gets trapped in the first local minimum encountered. Many optimization techniques such as Dynamic Hill Climbing [24], simulated annealing [21] and Tabu Search [10], which have been applied successfully in solving diverse optimization problems [21, 2, 9, 10, 24], have been used to overcome the main problem of simple hill-climbing or gradient-descent algorithms, and do not get trapped in local optima of the function being optimized. However, those algorithms are not guaranteed to find the global optima quickly, and may wander all around the search space. Directed Search (DS) overcomes the main problems of those approaches and effectively directs the search toward the received vector, which provides a global guidance to the search while escaping from local minima.

When the decoding problem is transformed into a continuous optimization problem, we attempt to minimize the cost function f_ϕ, i.e., to find a codeword in C' that is closest to the received vector ϕ. In DS the search is guided by two functions, f_ϕ and d_E, where f_ϕ is the

cost function and d_E is the Euclidean distance function. The Euclidean distance function, d_E, is used to explore regions successively closer to the received vector, and the cost function f_ϕ is used to guide the search for local minima in the regions explored. The procedure keeps an upper bound (UB) on the minimum values of f_ϕ calculated so far by the procedure. Every time the procedure finds a new local minimum, it generates the codeword corresponding to this local minimum and UB is updated if necessary.

The choice of the initial vector in every iteration may be made using the history as well as the information provided by the received vector and the inherent properties of the code, attempting to begin from a "good" part of the search space. In the main loop of the procedure, the search is first guided by the cost function f_ϕ, and a small modification is repeatedly made to the current vector, $(x_0, x_1, \ldots, x_{k-1}) \in R^k$, to move the vector closer to a local minimum, thus reducing the values of the continuous optimization function f_ϕ. If a local minimum is reached, the codeword with respect to this local minimum is generated and UB is updated if necessary. We note that DS will find a local minimum of the cost function, f_ϕ, before this loop stops since f_ϕ is bounded for any linear block codes.

In order to escape from the local minimum of f_ϕ and progress along the direction towards the received vector in R^n, the search is then guided by the Euclidean distance function d_E. The Euclidean distance between the received vector and the real vector $\boldsymbol{x} = (x_0, x_1, \ldots, x_{k-1}, \ldots, x_{n-1}) \in R^n$ is also calculated, where $(x_0, x_1, \ldots, x_{k-1}) \in R^k$ is the vector examined and the last $n-k$ components of $\boldsymbol{x}$ are obtained according to equation (1). A small modification is repeatedly made to the current vector, $(x_0, x_1, \ldots, x_{k-1}) \in R^k$, to move the vector whose corresponding vector in R^n is closer to the received vector. This process continues until either the values of f_ϕ of the vectors visited in R^k start to decrease, or no further improvement on the distance between the received vector and the vectors examined in R^n can be made. The loop continues until no improvement can be made in this iteration. We note that except for the last loop of the current iteration, DS will explore a new region in the search space before this loop stops since both d_E and f_ϕ are bounded for any linear block codes.

A new iteration is then started with a different initial vector. The termination criterion

imposes an upper bound on the number of iterations. It may also depend on whether recent iterations have yielded any improvement in the quality of the best codeword found so far.

We note that every local minimum of the cost function f_ϕ defined in Section 2.2 is a vertex of the solid k-cube of length two centered at the origin and corresponds to a codeword in C'. However, a codeword in C' is an n-dimensional vector. Any algorithms working in k-dimensional space will have no control on the redundant $n-k$ components of the code. Furthermore, the decoding complexity grows exponentially, and there is no easy way to guarantee that a real vector in the n-dimensional space is a codeword unless we check it. DS takes advantage of properties of the code and successfully works on both k-dimensional and n-dimensional spaces to make the search more efficient.

6 Hybrid Decoding Algorithm

In this section we describe a new decoding algorithm that hybridizes MA^*, H^*, and DS. This hybrid algorithm takes advantage of the strengths of these decoding algorithms and makes decoding more efficient than with the original MA^*, H^*, and DS decoding algorithms.

When the decoding problem is converted into a problem of graph-search through a code tree, we are interested in finding an optimal path from the start node to a goal node in the code tree of the transmitted code. The corresponding codeword of an optimal path in the code tree is a minimum-cost codeword when the decoding problem is transformed into a minimum-cost problem among codewords of the transmitted code. Furthermore, as described in Section 2.2, the minimum-cost codeword also corresponds to a codeword in C' that is closest in Euclidean distance to the received vector when the decoding problem is transformed into a continuous optimization problem.

The hybrid algorithm starts by finding a codeword whose cost is the initial upper bound UB of the decoding algorithm. A suboptimal version of the H^* algorithm is then applied to determine the values of some components among the first k most reliable, linearly independent components of a minimum-cost codeword. For every index $i \in \{0, 1, \ldots, k-1\}$ such that the value of component i of the minimum-cost codeword is still undetermined, the

suboptimal H^* generates at most p binary n-tuple vectors in $T''(i)$ to estimate the cost of a minimum-cost codeword, where p is a fixed positive integer. If some components of the minimum-cost codeword are determined, the code tree is pruned and the search space is reduced. MA^* is then applied to search for an optimal path through this subtree. During the decoding process, when MA^* finds a codeword whose cost is within a threshold Δ of UB, DS is applied to search for an even better upper bound, and the suboptimal H^* is again applied to determine the values of more components of a minimum-cost codeword. This process repeats until list $\mathcal{L}$ in MA^* is empty or the values of the first k components of a minimum-cost codeword have been determined by the suboptimal H^*. The hybrid decoding algorithm is described below, where we assume that $\boldsymbol{c}^* = (c_0^*, c_1^*, \ldots, c_{n-1}^*)$ is a minimum-cost codeword in $\boldsymbol{C}$.

Hybrid decoding algorithm

1. Let p and q be two positive integers and Δ a real number, $\Delta \geq 0$. Generate a nonempty set of codewords $S \subset \boldsymbol{C}$. For every codeword $\boldsymbol{c} \in S$, set $\boldsymbol{c}_{seed} = \boldsymbol{c}$ and calculate $low_q(m_{-1})$. Select the codeword $\boldsymbol{c} \in S$ that maximizes $low_q(m_{-1})$ as the initial seed of the decoding algorithm.

2. For every codeword in S, calculate the cost of the corresponding path from the start node to a goal node in the code tree. Let $\overline{\boldsymbol{c}}$ be the codeword in S that corresponds to the minimum-cost path $\boldsymbol{P}$ among all paths corresponding to codewords in S. Let UB be the cost of $\boldsymbol{P}$.

3. Apply DS with $((-1)^{\overline{c}_0}, (-1)^{\overline{c}_1}, \ldots, (-1)^{\overline{c}_{k-1}})$ as the initial vector. Set $\overline{\boldsymbol{c}}$ to be the codeword returned by DS and update UB if necessary. Set list $\mathcal{L} = \emptyset$.

4. Set $\overline{I} = \emptyset$ to be the set that consists of all indices t such that c_t^* have been determined. Let $I = \{0, 1, \ldots, k-1\} - \overline{I}$ be the set of indices i, $0 \leq i < k$, such that c_i^* is still undetermined. Apply the suboptimal H^* to determine the values of components of $\boldsymbol{c}^*$. For every $i \in I$, the suboptimal H^* generates at most p binary n-tuple vectors in $T''(i)$.

If the values of the first k components of $\boldsymbol{c}^*$ have been determined, the minimum-cost codeword is $(c_0^*, c_1^*, \ldots, c_{k-1}^*)\ \boldsymbol{G}$, and the algorithm terminates.

5. Apply MA^* to search for an optimal path through the pruned code tree of the transmitted code.

   (a) Let m be the node with least f_q value in list $\mathcal{L}$. Assume that node m is at level s in the pruned code tree. Let $\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_s$ be the labels of the path $\boldsymbol{P}_m$ from the start node to node m. If there exists an index t such that $\overline{v}_t \neq c_t^*$, $t \leq s$ and $t \in \overline{I}$, then this node can be discarded before it is expanded by MA^*.

   (b) For the codewords $\boldsymbol{c}'$ and $\boldsymbol{c}''$ generated at step 6 in the algorithm of MA^*, calculate $d_E(\boldsymbol{\phi},\ \boldsymbol{c}')$ and $d_E(\boldsymbol{\phi},\ \boldsymbol{c}'')$. If $d_E(\boldsymbol{\phi},\ \boldsymbol{c}') - d_E(\boldsymbol{\phi},\ \overline{\boldsymbol{c}}) \leq \Delta$, then DS is applied with $((-1)^{c_0'}, (-1)^{c_1'}, \ldots, (-1)^{c_{k-1}'})$ as the initial vector and UB is updated if necessary. If $d_E(\boldsymbol{\phi},\ \boldsymbol{c}'') - d_E(\boldsymbol{\phi},\ \overline{\boldsymbol{c}}) \leq \Delta$, then DS is applied with $((-1)^{c_0''}, (-1)^{c_1''}, \ldots, (-1)^{c_{k-1}''})$ as the initial vector, and UB is updated if necessary.

   (c) If the hybrid algorithm has found a better upper bound, the suboptimal H^* is applied again. For all $i \in I$ such that c_i^* is determined by the suboptimal H^* during this decoding process, $\overline{I}$ is set to be $\overline{I} \cup \{i\}$. If $\overline{I} = \{0, 1, \ldots, k-1\}$, then $(c_0^*, c_1^*, \ldots, c_{k-1}^*)\ \boldsymbol{G}$ is the minimum-cost codeword and the algorithm terminates; otherwise, MA^* continues to search for an optimal path in the pruned code tree until list $\mathcal{L}$ is empty.

We note that, during the decoding, every time the suboptimal H^* determines the value of a component of $\boldsymbol{c}^*$, the code tree is halved. Since the suboptimal H^* is invoked only when the algorithm finds a better upper bound, we may set $p \gg q$ without substantially increasing the time complexity of the algorithm. We also note that the hybrid algorithm is an MLSD decoding algorithm if no restriction is placed on the size of list $\mathcal{L}$ in MA^* in step 5 of the hybrid algorithm. However, we may limit the search by setting an upper bound on the size of list $\mathcal{L}$ in MA^*, as described in Section 3. In this case the hybrid algorithm is an SOSD

decoding algorithm. Simulation results for the hybrid MLSD and SOSD decoding algorithms are given in the next section.

7 Simulation Results for the AWGN Channel

We present simulation results for our hybrid MLSD and SOSD decoding algorithms for the $(128, 64)$, the $(256, 131)$, and the $(256, 139)$ binary-extended BCH codes transmitted over AWGN channels. As described in Section 6, if we don't limit the size of list $\mathcal{L}$ in MA^*, our hybrid algorithm is an MLSD decoding algorithm; otherwise, it is an SOSD decoding algorithm. We emphasize that the decoding algorithms presented in this paper are suitable for decoding any linear block codes. For linear block codes with length smaller than 128 and dimension smaller than 64, such as the $(104, 52)$ binary-extended quadratic residue code, MA^* achieves near-optimal to optimal performance for any signal-to-noise ratio value, as we have presented in our previous work [29].

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 $c_j = (-1)^{c_j}\sqrt{E}$ and $r_j = (-1)^{c_j}\sqrt{E} + e_j$, 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 N_0 per hertz. 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 use the SNR per transmitted information bit $\gamma_b = E_b/N_0 = \gamma n/k$ in our simulation. For AWGN channels, $\boldsymbol{\phi} = \frac{4\sqrt{E}}{N_0}\,\boldsymbol{r}$ [19], so we can substitute $\boldsymbol{r}$ for $\boldsymbol{\phi}$ in our decoding algorithm.

During the simulation we incorporate the criterion introduced in [31] to check if the minimum-cost codeword has been found and no further decoding is necessary.

Since we do not know the HW for each of these codes, we use a superset for it. Even though we use supersets of the HW for these codes, the following simulation results show that our hybrid algorithms still achieve optimal or near-optimal performance. We believe our hybrid algorithms will be able to decode longer codes with similar performance if we have better knowledge concerning the weight distribution of the code, which is another interesting

topic of research.

Since we do not know the HW for the $(128, 64)$ binary-extended BCH code, we use a superset for it. We know that $d_{\min} = 22$ for the $(128, 64)$ code and that the Hamming weight of any codeword is divisible by 2 [22]. Thus for this code the superset used is $\{x|$ (x is even and $22 \leq x \leq 106$) or ($x = 0$) or ($x = 128$)$\}$.

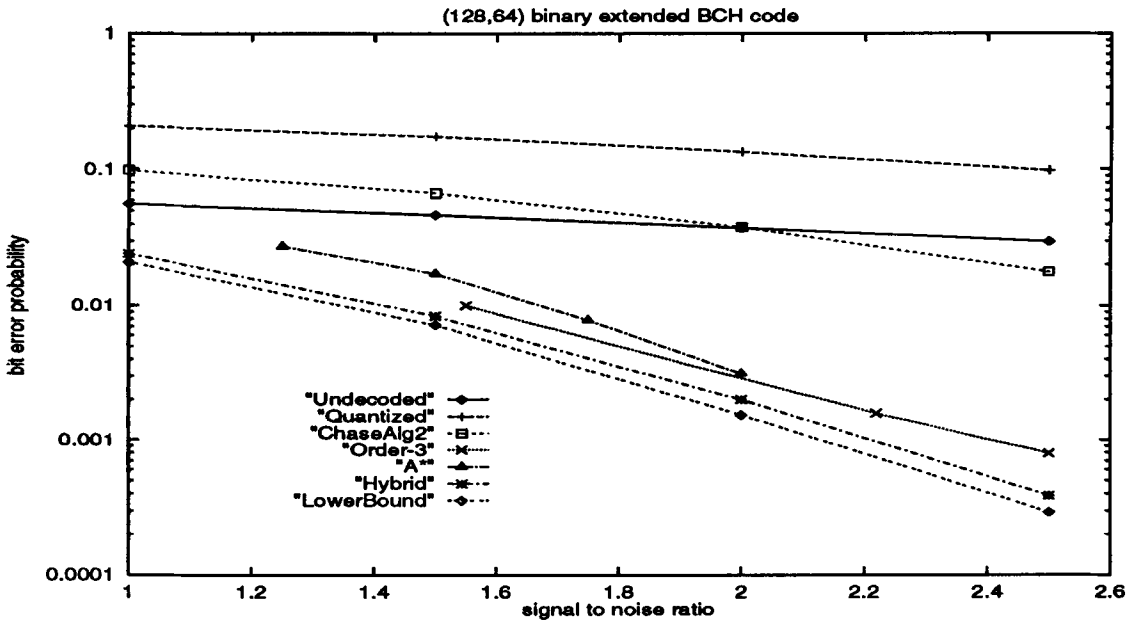

Figure 1: Comparison of the hybrid SOSD decoding algorithm with other algorithms for the $(128, 64)$ code

γ_b	1.0 dB		1.5 dB		2.0 dB		2.5 dB	
	max	ave	max	ave	max	ave	max	ave
N(r)	201088	108680.766	190484	92844.945	203610	65945.297	225437	41528.496
C(r)	114661	45644.414	106641	36529.672	107812	23945.756	120457	13785.361
M(r)	10000	8818.749	10000	7687.217	10000	5611.436	10000	3605.018
p	256		256		128		128	

Table 1: Performance of the hybrid SOSD decoding algorithm for the $(128, 64)$ code

Simulation results of our hybrid SOSD decoding algorithm for the $(128, 64)$ code for $\gamma_b =$ 1.0 dB, 1.5 dB, 2.0 dB, and 2.5 dB are given in Figure 1 (Hybrid) and Table 1, where q is

set to be 4 and $M_B = 10,000$. Furthermore, we set $\Delta = 0$ as our threshold to apply DS in our simulation. The following notations are used in the tables:

$\gamma_b = E_b/N_0$,

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

$\mathrm{C}(\boldsymbol{r})$ = number of codewords constructed during the decoding of $\boldsymbol{r}$,

$\mathrm{M}(\boldsymbol{r})$ = number of nodes stored in list $\mathcal{L}$ during the decoding of $\boldsymbol{r}$,

max = maximum value among samples tried,

ave = average value among samples tried, and

p = the maximum number of vectors generated by H^* in order to fix the value of one of the first k components during the decoding process.

During the decoding process, $\mathrm{N}(\boldsymbol{r})$ nodes in the code tree are checked to see if they should be opened, expanded, or deleted by the algorithm. Even though the algorithm visits $\mathrm{N}(\boldsymbol{r})$ number of nodes, it actually stores no more than $\mathrm{M}(\boldsymbol{r})$ nodes and tries only $\mathrm{C}(\boldsymbol{r})$ codewords which are visited either by MA^* or DS during the decoding process.

For the $(128, 64)$ code (Figure 1), the performance of our hybrid SOSD decoding algorithm is within 0.10 dB of the lower bound of the performance of the MLSD decoding algorithm. Thus, for the samples tried, limiting the size of list $\mathcal{L}$ to $10,000$ nodes introduced only a small degradation on the performance of the algorithm for the $(128, 64)$ code. Furthermore, for the samples tried, the average number of codewords constructed is approximately 15 orders of magnitude smaller than the total number of codewords, ($2^{64} \simeq 1.85 \times 10^{19}$), and in the worst case the maximum number of codewords constructed is approximately 14 orders of magnitude smaller than the total number of codewords.

For comparison purposes (Figure 1), we also give the bit error probability of the quantized received vector (Quantized) and an estimate of the performance of Chase Algorithm 2

(ChaseAlg2) [3]. The "Quantized" data is obtained as follows: we quantize the k most reliable, linearly independent components of the received vector and obtain a k-tuple, which is then multiplied with the generator matrix $\boldsymbol{G}$ to obtain a codeword which is returned as the result of the decoding process. It would be impractical to simulate Chase Algorithm 2, because there is no practical decoder for the $(128, 64)$ code that corrects ten digital errors. The estimate of Chase Algorithm 2 was obtained as follows: for a given $\boldsymbol{r}$, if the number of errors in the $(n - \lfloor (d_{\min}/2) \rfloor)$ most reliable positions is less than or equal to $\lfloor (d_{\min} - 1)/2 \rfloor$, then we will assume there is no decoding error, since the transmitted codeword would be generated in the decoding procedure of $\boldsymbol{r}$. Otherwise, the decoder output is the quantized version of $\boldsymbol{r}$. A lower bound on the bit error probability of the MLSD decoding algorithm (LowerBound) [5] and the bit error probability for undecoded data (Undecoded) are also given.

In addition, we also give the simulation results of the A^* decoding algorithm provided in [15] (A^*) and simulation results of the "Order-3" decoding procedure provided in [7] (Order-3). As indicated in Figure 1, the performance of our hybrid SOSD decoding algorithm is better than those of the Chase Algorithm 2 (ChaseAlg2), A^* algorithm (A^*), and the "Order-3" decoding procedure (Order-3).

We note that in [15] simulation results of the A^* decoding algorithm are not provided for 2.5 dB $\leq \gamma_b \leq$ 4.5 dB, since the performance of the A^* algorithm deteriorates substantially for those ranges of signal-to-noise ratio. In [7], simulation results of the error performance for the "Order-3" decoding procedure for this code are provided for 2.22 dB $\leq \gamma_b \leq$ 3.47 dB. For 3.47 dB $< \gamma_b \leq$ 5.23 dB, only Union bounds on the error probability are provided. As pointed out in [7], the number of simulated samples are far too small to obtain reliable information for those ranges of signal-to-noise ratio. In [7] it is also shown that the error performance of the "Order-4" decoding procedure is similar to the performance of our hybrid SOSD algorithm for 1.55 dB $\leq \gamma_b \leq$ 2.5 dB; however, both the maximum and the average number of computations of the "Order-4" decoding procedure are enormous. For example, for $\gamma_b = 2.5$ dB the maximum number of computations of the "Order-4" decoding processing (N_{max4} in [7]) is equal to $43,464,512$, and the average number of computations of the "Order-

4" decoding processing (N_{ave4} in [7]) is approximately $10^{6.5}$ ($\simeq 3.16 \times 10^6$). Very recently some improvements to this decoding process were proposed to allow computation savings or decoding speedup with little error performance degradation [8]. However, in order to obtain near-optimal performance for the (128, 64) code, the modified decoding procedure still requires a tremendous number of computations ($N_{max} = 34,932,480$) [8].

γ_b	3 dB		4 dB		5 dB		6 dB		7 dB		8 dB	
	max	ave	max	ave	max	ave	max	ave	max	ave	max	ave
N(r)	206961	19725.561	155333	1697.749	123027	38.377	2081	0.796	379	0.047	0	0.000
C(r)	167235	6046.271	77230	412.087	43504	8.873	133	1.439	57	1.190	2	1.123
M(r)	10000	1727.780	10000	150.837	10000	4.796	380	0.211	50	0.010	0	0.000
p	64		32		8		4		4		4	

Table 2: Performance of the hybrid SOSD decoding algorithms for the (128, 64) code

Simulation results of our hybrid SOSD decoding algorithm for the (128, 64) code for γ_b equal to 3.0 dB, 4.0 dB, 5.0 dB, 6.0 dB, 7.0 dB, and 8.0 dB are given in Table 2. These results were obtained by simulating 35,000 samples for each SNR when q is set to be 4 and M_B is set to be 10,000. Furthermore, we set Δ equal to 20 as our threshold to apply DS in our simulation for γ_b equal to 3 dB. Among the 35,000 samples tried we found one sample such that the decoded codeword is closer to the received vector than to the transmitted codeword. No other decoding error occurred during simulation. For the other cases, i.e., 4.0 dB $\leq \gamma_b \leq$ 8.0 dB, we set $\Delta = 0$ as our threshold to apply DS. No decoding error occurred during simulation for these cases.

We note that for the (128, 64) code with $\gamma_b \geq 6$ dB the memory bound M_B is never reached for the samples tried. In this case the performance of our hybrid algorithm is the same as that of the MLSD decoding algorithm.

Since we do not know the HW for the (256, 131) binary-extended BCH code, we use a superset for it. We know that $d_{\min} = 38$ and that the Hamming weight of any codeword is divisible by 2. Thus for this code the superset used is $\{x|$ (x is divisible by 2 and $38 \leq x \leq 218$), or ($x = 0$), or ($x = 256$)$\}$.

γ_b	5 dB		6 dB		7 dB		8 dB	
	max	ave	max	ave	max	ave	max	ave
N($\boldsymbol{r}$)	351717	274.215	4786	4.743	1750	0.407	0	0
C($\boldsymbol{r}$)	105828	40.764	197	1.992	119	1.251	3	1.115
M($\boldsymbol{r}$)	15000	37.353	2765	2.320	625	0.093	0	0
p	16		8		8		8	

Table 3: Performance of the hybrid SOSD decoding algorithms for the $(256, 131)$ code

The simulation results of our hybrid SOSD decoding algorithm for the $(256, 131)$ code for γ_b equal to 5.0 dB, 6.0 dB, 7.0 dB, and 8.0 dB are given in Table 3, where q is set to be 8 and M_B is set to be $15,000$. Furthermore, we set $\Delta = 0$ as our threshold to apply DS in our simulation. These results were obtained by simulating 35,000 samples for each SNR. No decoding error occurred during simulation.

We note that simulation results show that for $\gamma_b \geq 6$ dB the memory bound M_B is never reached for the samples tried. In this case the performance of our hybrid algorithm is the same as that of the MLSD decoding algorithm. Both the maximum number of codewords and the average number of codewords constructed by our hybrid decoding algorithm are insignificant compared with the total number of codewords, ($2^{131} \simeq 2.72 \times 10^{39}$).

In order to compare our hybrid MLSD decoding algorithm with the MLSD decoder presented in [20], we also simulate our hybrid MLSD decoding algorithm for the $(256, 139)$ binary-extended BCH code. Since we do not know the HW for the $(256, 139)$, we use a superset for it. We know that $d_{\min} = 32$ and that the Hamming weight of any codeword is divisible by 2. Thus for this code the superset used is $\{x|$ (x is divisible by 2 and $32 \leq x \leq 224$), or ($x = 0$), or ($x = 256$)$\}$.

γ_b	5.5 dB		6.0 dB		6.5 dB		7.0 dB	
	max	ave	max	ave	max	ave	max	ave
N($\boldsymbol{r}$)	57635	19.644	4699	3.342	2763	0.892	1238	0.154
C($\boldsymbol{r}$)	4570	3.384	195	1.827	149	1.402	99	1.228
M($\boldsymbol{r}$)	5436	6.772	3016	1.549	998	0.257	153	0.037
p	16		16		8		8	

Table 4: Performance of the hybrid MLSD decoding algorithm for the (256, 139) code

The simulation results of our hybrid MLSD decoding algorithm for the (256, 139) code for γ_b equal to 5.5 dB, 6.0 dB, 6.5 dB, and 7.0 dB are given in Table 4, where q is set to be 8. These results were obtained by simulating 35,000 samples for each SNR. No decoding error occurred during simulation. Furthermore, for the examples tried, both the maximum number of codewords and the average number of codewords constructed by our hybrid MLSD decoding algorithm are insignificant compared with the total number of codewords, ($2^{139} \simeq 6.97 \times 10^{42}$).

Table 4 attests to the fact that the complexity of our hybrid MLSD decoding algorithm is much less than that of the MLSD decoder presented in [20]. For example, for γ_b equal to 5.5 dB the maximum number of codewords tried in [20] is 2,097,152, while the maximum number of codewords tried in our hybrid MLSD decoder is only 4,570, which is about three orders of magnitude better in maximum decoding complexity; the average number of codewords tried in [20] is 76.21, while the average number of codewords tried in our hybrid MLSD decoding algorithm is only 3.384, which is approximately 20 times better in average decoding complexity. Furthermore, as shown in Table 4, the memory requirement of our hybrid MLSD decoding algorithm is small for the (256, 139) code for γ_b greater than or equal to 5.5 dB.

8 Conclusion

We have proposed three decoding techniques and a hybrid of these techniques. MA^* is a graph-search algorithm that originates from A^* in Artificial Intelligence [18, 26]. Simulation results have shown that MA^* is very efficient and has near-optimal to optimal performance for codes of short-to-moderate lengths [28], but becomes impractical for codes of long lengths transmitted over channels with low SNR, since the memory requirement is large. H^* is a new algorithm that is very flexible and can be incorporated with other techniques to reduce the search space during the decoding process. For example, simulation results for the (128,64) code transmitted over channels with $\gamma_b \geq 6$ dB show that all the values of the first k components of a minimum-cost codeword are determined by H^* with p equal to 4 for the 35000 samples tried during the decoding process, and the search space is reduced to the minimum, i.e. from 2^{64} to $2^{64-64} = 1$. However, for long codes transmitted over channels with low SNR, the number of vectors generated to fix the value of a component of the minimum-cost codeword grows exponentially.

DS is a novel heuristic approach that explores the search space between a given vector and the received vector. It directs the search toward the received vector while escaping from a local minimum. In *DS* the search is guided by global information provided by the received vector. The beauty of this approach is that it takes advantage of properties of the code and successfully works on both k-dimensional and n-dimensional spaces and efficiently finds the best solution in the space between a real vector and the received vector. Furthermore, *DS* requires no memory. It is simple to implement and can be incorporated with other decoding algorithms.

The hybrid algorithm takes advantage of the strengths of these three algorithms and makes the decoding more efficient. The strength of H^* can overcome the weakness of MA^*, and vice versa. For codes of long lengths transmitted over low SNR, the memory requirement of MA^* is impractical, while the memory requirement of H^* is small. Furthermore, the number of vectors that must be generated before H^* can fix the value of a component of the minimum-cost codeword depends on how close the upper bound is to the received vector, and

MA^* gives this upper bound during decoding process. H^* works well with MA^*, since the tendency of MA^* is to find successfully better upper bounds during the decoding process. DS is applied to obtain an even better upper bound when MA^* explores a new region in the search space. It efficiently finds the closest codeword to the received vector in the region between the received vector and a codeword obtained by MA^*. Simulation results show that this hybrid algorithm can efficiently decode the $(128, 64)$ binary-extended BCH code for any signal-to-noise ratio value and has near-optimal to optimal performance. Previously, no practical decoder could have decoded this code with such a performance for all these ranges of signal-to-noise ratio. We can also decode very efficiently the $(256, 131)$ and the $(256, 139)$ binary-extended BCH codes for signal-to-noise ratio (γ_b) greater than or equal to 5 dB. We emphasize that for most practical communication systems the probability of error is less than 10^{-3} (γ_b greater than 6.8 dB).

We would like to emphasize here the flexibility of this decoding algorithm. For example, (1) it is applicable to any linear block code; (2) it does not require the availability of a hard decision decoder; (3) in order to make it more efficient to decode a particular code, we can design heuristic functions that take advantage of the specific properties of this code; (4) any termination criterion can be easily incorporated into it. For example, the criterion proposed in [31] can be incorporated to determine the termination condition of our decoding algorithm.

In addition, we would like to point out that the algorithms presented in this paper are suitable for parallel implementations. MA^* can be parallelized analogously to Algorithm A^*, in which a very good speed-up was obtained [11]. H^* can be parallelized easily, since the heuristic functions for the k most reliable, linearly independent components of a minimum-cost codeword can be calculated simultaneously and independently. DS can also be parallelized, since the generation of real vectors reachable from a current vector can be implemented independently. This will substantially reduce 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 hybrid algorithm.

Note that we used only one seed for computing the heuristic functions h_q, h'_q, and h''_p.

However, we can generalize the procedure to calculate function h with respect to several seeds. Details of this approach can be found in [12].

We note that even though the heuristic functions we use in MA^* and H^* depend on the weight distribution of the code, which may remain unknown, our hybrid algorithm still achieves optimal or near-optimal performance when we use supersets of the HW for the codes with length less than or equal to 256. We believe our hybrid algorithm will be able to decode longer codes with similar performance if we have better knowledge concerning the weight distribution of the code, which is another interesting topic of research. Furthermore, the performance of the H^* algorithm depends on the heuristic functions and the quality of UBs obtained during the decoding process. In this paper we have focused on the design of algorithms. The performance analysis of the H^* algorithm remains to be studied.

We also note that DS is a heuristic approach. In future work, we plan to work out a general theory of the approach, and examine its complexity and performance.

Another important line of research could be to investigate the use of other optimization techniques to design efficient SOSD decoding algorithms for long block codes. Optimization techniques such as simulated annealing [21], Tabu Search [10], and genetic algorithms [23] have been used with great success in important practical applications. These optimization techniques may be modified to take into consideration the properties of linear block codes and applied to the design of efficient SOSD decoding algorithms for long block codes.

Appendices

Appendix A — The Optimality of MA^*

In this appendix we show the optimality of MA^* and a corollary used in the design of the algorithm. Theorem 1 guarantees that MA^* still finds an optimal path if it discards those nodes whose values of low_q are greater than or equal to UB. Corollary 1 shows that if a node is discarded according to Theorem 1, then the optimal path that goes through this node must have a cost greater than or equal to UB. Therefore, Corollary 2 shows that from the start node to a goal node along a path in the code tree, the values of the low_q are monotonically non-decreasing, i.e., if m_s is an immediate successor of node m_{s-1}, then $low_q(m_{s-1}) \leq low_q(m_s)$. It follows that if a node is discarded according to Theorem 1, then all successors of this node must have values of low_q greater than or equal to UB and can be discarded. Note that Theorem 1 and Corollary 2 are used in the design of our decoding algorithm to reduce the size of list $\mathcal{L}$. In order to show the proofs of the theorem and corollaries, we first give the following lemmas. Corollary 2 follows directly from Lemma 4. The proofs of Theorem 1 and Corollary 1 are also given in this appendix.

Lemma 1

Let m be a node in the code tree. For any two positive integers i and j, if $i < j$, then $h'_i(m) \leq h'_j(m)$.

Proof of Lemma 1:

Let $\boldsymbol{v}'_i(m) \in T'_i(m)$ be the vector used to calculate $h'_i(m)$ and let $\boldsymbol{v}'_j(m) \in T'_j(m)$ be the vector used to calculate $h'_j(m)$. According to the definitions of $h'_i(m)$ and $h'_j(m)$, since $i < j$, it follows that $T'_i(m) \supset T'_j(m)$, which implies that $h'_i(m) \geq h'_j(m)$.

Lemma 2

For a positive integer q and for any node m in the code tree,

$$h'_q(m) \leq h^*(m),$$

where $h^*(m)$ is the actual cost of a minimum-cost path from node m to a goal node.

Proof of Lemma 2:

We prove by contradiction that $h'_q(m) \leq h^*(m)$. Let m be a node at level l and let $\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_l$ be the labels of the path $\boldsymbol{P}_m$ from the start node to node m. Let $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$ be the codeword that corresponds to the labels of the minimum-cost path from the start node to a goal node that goes through node m, i.e., $\boldsymbol{c} \in \boldsymbol{C}$ and $c_0 = \overline{v}_0, c_1 = \overline{v}_1, \ldots, c_l = \overline{v}_l$. Then, $\boldsymbol{c} \in T'(m)$ and $h^*(m) = \sum_{j=0}^{n-1} (\phi_j - (-1)^{c_j})^2$. Assume that $h'_q(m) > h^*(m)$. Let's consider two cases, namely $q = 1$ and $q > 1$.

1. Case 1: $q = 1$. According to the definition of $h'_q(m)$, if $q = 1$, then $h'_1(m) = \min_{\boldsymbol{v}' \in T'_1(m)} \left\{ \sum_{j=0}^{n-1} \left(\phi_j - (-1)^{v'_j}\right)^2 \right\} > h^*(m) = \sum_{j=0}^{n-1} (\phi_j - (-1)^{c_j})^2$ which is a contradiction that $\boldsymbol{c} \in T'_1(m)$;

2. Case 2: $q > 1$. If $q > 1$, then according to the definition of $h'_q(m)$ it follows from the assumption that $h'_q(m) > h^*(m)$ and from Lemma 1 that there exists a positive integer i, $1 \leq i < q$ such that $h'_i(m) = h^*(m)$. Since the vector used to calculate $h'_i(m)$ is the codeword $\boldsymbol{c}$, by the definition of $h'_q(m)$, $h'_q(m) = h'_i(m) = h^*(m)$, which is also a contradiction to the assumption that $h'_q(m) > h^*(m)$.

Lemma 3

For a positive integer q, function h_q is monotonic [26], i.e., for any integer s, $0 \leq s < n$, if node m_s at level s is an immediate successor of node m_{s-1} in the code tree, then

$$h_q(m_{s-1}) \leq h_q(m_s) + c(m_{s-1}, m_s), \tag{2}$$

where $c(m_{s-1}, m_s)$ is the arc cost between node m_{s-1} and node m_s.

Proof of Lemma 3:

Let $\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_{s-1}$ be the labels of the path $\boldsymbol{P}_{m_{s-1}}$ from the start node to node m_{s-1}. Let c_s be the label of the arc from node m_{s-1} to node m_s. Then $c(m_{s-1}, m_s) = (\phi_s - (-1)^{c_s})^2$. We will prove that $h_q(m_{s-1}) \leq c(m_{s-1}, m_s) + h_q(m_s)$ by considering the following three cases:

1. $0 \leq s < k-1$:

   Let $\boldsymbol{v}_q = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_{s-1}, c_s, \ldots, v_{q(n-1)}) \in T_q(m_s)$ be the vector that satisfies

   $$\sum_{j=s+1}^{n-1} (\phi_j - (-1)^{v_{qj}})^2 = \min_{\boldsymbol{v} \in T_q(m_s)} \left\{ \sum_{j=s+1}^{n-1} (\phi_j - (-1)^{v_j})^2 \right\}.$$

   Since $(\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_{s-1}, c_s, v_{q(s+1)}, \ldots, v_{q(n-1)}) \in T_q(m_{s-1})$, it follows that

   $$\min_{\boldsymbol{v} \in T_q(m_{s-1})} \left\{ \sum_{j=s}^{n-1} (\phi_j - (-1)^{v_j})^2 \right\} \leq c(m_{s-1}, m_s) + \min_{\boldsymbol{v} \in T_q(m_s)} \left\{ \sum_{j=s+1}^{n-1} (\phi_j - (-1)^{v_j})^2 \right\}.$$

2. $s = k-1$:

   Since $h_q(m_{s-1}) \leq h^*(m_{s-1})$ and $h_q(m_s) = h^*(m_s)$, it follows that $h_q(m_{s-1}) \leq h^*(m_{s-1})$ $\leq c(m_{s-1}, m_s) + h^*(m_s) = c(m_{s-1}, m_s) + h_q(m_s)$.

3. $k-1 < s < n$:

   Since $h_q(m_{s-1}) = h^*(m_{s-1})$ and $h_q(m_s) = h^*(m_s)$, it follows that $h_q(m_{s-1}) = h^*(m_{s-1})$ $\leq c(m_{s-1}, m_s) + h^*(m_s) = c(m_{s-1}, m_s) + h_q(m_s)$.

Lemma 4

For a positive integer q, function h'_q is also monotonic, i.e., for any integer s, $0 \leq s < n$, if node m_s at level s is an immediate successor of node m_{s-1} in the code tree, then

$$h'_q(m_{s-1}) \leq h'_q(m_s) + c(m_{s-1}, m_s), \tag{3}$$

where $c(m_{s-1}, m_s)$ is the arc cost between node m_{s-1} and node m_s.

Proof of Lemma 4:

Let $\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_{s-1}$ be the labels of the path $\boldsymbol{P}_{m_{s-1}}$ from the start node to node m_{s-1}. Let c_s be the label of the arc from node m_{s-1} to node m_s. Then, $c(m_{s-1}, m_s) = (\phi_s - (-1)^{c_s})^2$. According to the definitions of $S_H(m_{s-1})$ and $S_H(m_s)$, we have $S_H(m_{s-1} \subseteq S_H(m_s)$. The proof that $h'_q(m_{s-1}) \leq c(m_{s-1}, m_s) + h'_q(m_s)$ when $S_H(m_{s-1}) = S_H(m_s)$ is similar to that of Lemma 3. We now prove that $h'_q(m_{s-1}) \leq c(m_{s-1}, m_s) + h'_q(m_s)$ when $S_H(m_s) - S_H(m_{s-1}) \neq \emptyset$. Without loss of generality, we may assume that $\boldsymbol{h}_l$ is the only parity check in $S_H(m_s) -$

$S_H(m_{s-1})$. Let $\boldsymbol{v}'_q = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_{s-1}, c_s, v'_{q(s+1)}, \ldots, v'_{q(n-1)}) \in T'_q(m_s)$ be the vector that satisfies that

$$\sum_{j=s+1}^{n-1} \left(\phi_j - (-1)^{v'_{qj}}\right)^2 = \min_{\boldsymbol{v}' \in T'_q(m_s)} \left\{ \sum_{j=s+1}^{n-1} \left(\phi_j - (-1)^{v'_j}\right)^2 \right\};$$

then $v'_{q(k+l)} = (\overline{v}_0 h_{l0} \oplus \overline{v}_1 h_{l1} \oplus \ldots \oplus \overline{v'}_{s-1} h_{l(s-1)} \oplus c_s h_{ls})$. Since $\boldsymbol{v}'_q = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_{s-1}, c_s, v'_{q(s+1)}, \ldots, v'_{q(k+l)}, \ldots, v'_{q(n-1)})$ is also in $T'_q(m_{s-1})$, it follows that

$$\min_{\boldsymbol{v}' \in T'_q(m_{s-1})} \left\{ \sum_{j=s}^{n-1} \left(\phi_j - (-1)^{v'_j}\right)^2 \right\} \leq c(m_{s-1}, m_s) + \min_{\boldsymbol{v}' \in T'_q(m_s)} \left\{ \sum_{j=s+1}^{n-1} \left(\phi_j - (-1)^{v'_j}\right)^2 \right\}.$$

Theorem 1

Let $\boldsymbol{P}$ be a path from the start node to a goal node found by MA^*. Let UB be the cost of $\boldsymbol{P}$. MA^* still finds an optimal path if it discards from the code tree any node m for which $low_q(m) \geq UB$.

Proof of Theorem 1:

Let $\boldsymbol{P}^* = (m_{-1}, m_0^*, \ldots, m_l^*, \ldots, \ldots, m_{n-1}^*)$ be an optimal path. Let m_l^* be the first node in path $\boldsymbol{P}^*$ that is in list $\mathcal{L}$. According to Lemma 1,

$$low_q(m_l^*) = g(m_l^*) + h'_q(m_l^*) \leq g(m_l^*) + h^*(m_l^*) = f^*(m_l^*),$$

where $f^*(m_l^*) = f^*(m_{n-1}^*)$ is the cost of the optimal path. It follows that

$$low_q(m_l^*) \leq f^*(m_l^*) \leq UB.$$

Therefore, if $low_q(m_l^*) < UB$, node m_l^* will not be discarded; otherwise, if $low_q(m_l^*) = UB$, then the algorithm has already found an optimal path and node m_l^* can be discarded.

Corollary 1

Let m be a node in the code tree. For any positive integer q, if $low_q(m) \geq UB$, then the cost of the optimal path that goes through node m is greater than or equal to UB.

Proof of Corollary 1:

Let $f^*(m) = g^*(m) + h^*(m)$ be the actual cost of the optimal path that goes through node m, where $g^*(m)$ is the actual cost of the path from the start node to node m and $h^*(m)$ is the actual cost of the optimal path from node m to a goal node. Assume that $low_q(m) \geq UB$. This theorem follows directly from Lemma 2 and the definition of low_q since $f^*(m) = g^*(m) + h^*(m) = g(m) + h^*(m) \geq g(m) + h'(m) = low_q(m) \geq UB$.

Corollary 2

For a positive integer q, if m_i is an immediate successor of a node m_{i-1} in the code tree, then $low_q(m_i) \geq low_q(m_{i-1})$.

Proof of Corollary 2:

According to Lemma 4, $h'_q(m_{s-1}) \leq c(m_{s-1}, m_s) + h'_q(m_s)$. It follows that $low_q(m_{s-1}) = g(m_{s-1}) + h'_q(m_{s-1}) \leq g(m_{s-1}) + c(m_{s-1}, m_s) + h'_q(m_s) \leq g(m_s) + h'_q(m_s) = low_q(m_s)$.

Appendix B — Algorithm for the Calculation of Function $h_q(m)$

In this appendix we give an algorithm for the calculation of heuristic function h_q. The complexity of this algorithm is $O(qn + q\log(q))$. In the algorithm we assume that $\boldsymbol{\phi}' = (\phi'_0, \phi'_1, \ldots, \phi'_{n-1})$ is the received vector, which is sorted in nondecreasing order, i.e., $\phi'_{j_1} \leq \phi'_{j_2}$ if $j_1 < j_2$. Let $HW = \{w_i | 0 \leq i \leq J\}$ be the set of all $J+1$ distinct Hamming weights that codewords of $\boldsymbol{C}$ may have. Assume that w is the Hamming weight of the quantized received vector. Let $HEAP$ be a binary tree such that each node in $HEAP$ stores a vector in $\{0,1\}^n$ and the vector stored in each node of $HEAP$ is closer to $\boldsymbol{\phi}'$ than any of the vectors stored in its children. In this algorithm, "$\vec{1}^n$" represents a string of 1s whose length is greater than or equal to 0, i.e., $(\vec{1}) = ()$ or $(1, 1, \ldots, 1)$. Analogously, $(\vec{0}) = ()$ or $(0, 0, \ldots, 0)$.

1. Set $HEAP = \emptyset$ and $j = 0$. $O(1)$

2. Let $w_t \in HW$ be the largest Hamming weight smaller than or equal to w and let $w_{t+1} \in HW$ be the smallest Hamming weight greater than w. Assume that $\boldsymbol{v} = (1, 1, \ldots, 1, 0, \ldots, 0) \in \{0,1\}^n$ is the vector with Hamming weight w_t and $\boldsymbol{v}' = (1, 1, \ldots, 1, 0, \ldots, 0) \in \{0,1\}^n$ is the vector with Hamming weight w_{t+1}. If $d_E(\boldsymbol{\phi}', \boldsymbol{v}) \leq d_E(\boldsymbol{\phi}', \boldsymbol{v}')$, then the closest vector to $\boldsymbol{\phi}'$ in $\{0,1\}^n$ that satisfies the Hamming weight constraint is $\boldsymbol{v}$; otherwise, $\boldsymbol{v}'$ is the closest vector to $\boldsymbol{\phi}'$ in $\{0,1\}^n$ that satisfies the Hamming weight constraint. Without loss of generality, we may assume that $\boldsymbol{v}$ is the closest vector to $\boldsymbol{\phi}'$ that satisfies the Hamming weight constraint. Set $\boldsymbol{v}$ to be the first vector in group 1 with weight w_t, i.e., $\boldsymbol{p}_1(1, w_t) = \boldsymbol{v}$. Put $\boldsymbol{p}_1(1, w_t)$ into $HEAP$. Let n_g be the number of groups that have been created. Set $n_g = 1$. $2 \times O(n)$.

3. Heapify all vectors in $HEAP$ such that the vector stored in the root of $HEAP$ is the closest vector to $\boldsymbol{\phi}'$ among all vectors stored in $HEAP$. Let $\boldsymbol{p}_\alpha(\beta, w_i)$ be the vector stored in root of $HEAP$. Set $j = j + 1$. $O(\log(q))$

4. $\boldsymbol{p}_\alpha(\beta, w)$ is the j^{th} closest vector to $\boldsymbol{\phi}'$ that satisfies the Hamming weight constraint. If $j = q$, the closest q vectors to $\boldsymbol{\phi}'$ that satisfy the Hamming weight constraint have

been generated and the algorithm terminates; otherwise, set $\boldsymbol{v} = (v_0, v_1, \ldots, v_{n-1})$ to be $\boldsymbol{p}_\alpha(\beta, w_i)$. $O(n)$

5. If $\alpha > 1$, go to step 7. $O(1)$

6. If $w_{i-1} \in HW$, set $n_g = n_g + 1$ and generate the first vector of group n_g that is the closest vector to $\boldsymbol{\phi}'$ with weight w_{i-1}. Put $\boldsymbol{p}_1(n_g, w_i)$ into $HEAP$. If $w_{i+1} \in HW$, set $n_g = n_g + 1$ and generate the first vector of group n_g that is the closest vector to $\boldsymbol{\phi}'$ with weight w_{i+1}. Put $\boldsymbol{p}_1(n_g, w_i)$ into $HEAP$. $2 \times O(n)$

7. If $\boldsymbol{p}_\alpha(\beta, w_i) = (\ldots, 0, 0, \vec{1}, 1)$, then delete $\boldsymbol{p}_\alpha(\beta, w_i)$ from $HEAP$ and go to step 3. $O(1)$

8. If $\boldsymbol{p}_\alpha(\beta, w_i) = (\ldots, 1, 0, \vec{1}, 1)$, then set $\boldsymbol{p}_\alpha(\beta, w_i) = (\ldots, 0, 1, \vec{1}, 1)$. Set $\alpha = \alpha + 1$. Put $\boldsymbol{p}_\alpha(\beta, w_i)$ into $HEAP$ and go to step 3. $O(1)$

9. Let v_s be the last component of $\boldsymbol{p}_\alpha(\beta, w_i)$ whose value is 1, i.e., $v_s = 1$ and $v_{s+1} = v_{s+2} =, \ldots, = v_{n-1} = 0$. $O(n)$

10. Shift the last 1 in $\boldsymbol{p}_\alpha(\beta, w_i)$ right to generate the next vector, $\boldsymbol{p}_{\alpha+1}(\beta, w_i)$, i.e., set $(v_s, v_{s+1}) = (0, 1)$ and $\alpha = \alpha + 1$. Put $\boldsymbol{p}_\alpha(\beta, w_i)$ into $HEAP$. $O(1)$

11. If $\boldsymbol{p}_\alpha(\beta, w_i)$ is equal to $(\ldots, 1, 0, \vec{1}, 0, 1, 0, \vec{0})$, then create a new group such that the vector $(\ldots, 0, 1, \vec{1}, 1, 0, 0, \vec{0})$ is the first vector in this group, i.e., set $n_g = n_g + 1$ and $\boldsymbol{p}_1(n_g, w_i) = (\ldots, 0, 1, \vec{1}, 1, 0, 0, \vec{0})$. Put $\boldsymbol{p}_1(n_g, w_i)$ into $HEAP$. Go to step 3. $O(1)$

The complexity of the algorithm is $O(qn + q\log(q))$.

Appendix C — The Optimality of H^*

In this appendix we show the optimality of the H^* algorithm. Let $\overline{c} = (\overline{c}_0, \overline{c}_1, \ldots, \overline{c}_{n-1})$ be the codeword whose cost is the upper bound UB and assume that $c^* = (c_0^*, c_1^*, \ldots, c_{n-1}^*)$ is a minimum-cost codeword. Let $\overline{I}$ be the set that contains all indices $t \in \{0, 1, \ldots, n-1\}$ such that the value of the t^{th} component of a minimum-cost codeword has been determined, i.e., c_t^* is already determined. Let $I = \{0, 1, \ldots, k-1\} - \overline{I}$ be the set of indices i in which c_i^* is still undetermined. For an index $i \in I$, Theorem 2 guarantees that H^* will find a minimum-cost codeword whose i^{th} component is $\overline{c}_i \oplus 1$. If the cost of this minimum-cost codeword is less than UB, the algorithm has found a codeword whose cost is less than UB and UB is updated; otherwise, the value of the i^{th} component of a minimum-cost codeword could be determined. Theorem 3 guarantees that if $low'_j(i, \overline{c}_i) \geq UB$, then the i^{th} component of the minimum-cost codeword is $\overline{c}_i$. These two theorems guarantee that H^* will find a minimum-cost codeword before it terminates. In order to show the proofs of these two theorems, we first give the following lemma.

Lemma 5

For an index $i \in I$ and any positive integers j_1 and j_2, if $j_1 < j_2$, then $h''_{j_1}(i) \leq h''_{j_2}(i)$ and $low'_{j_1}(i, \overline{c}_i) \leq low'_{j_2}(i, \overline{c}_i)$.

Proof of Lemma 5:

Let $\boldsymbol{v}''_{j_1} = (v''_{j_1 0}, v''_{j_1 1}, \ldots, v''_{j_1 (n-1)})$ be the vector used to calculate $h''_{j_1}(i)$ and let $\boldsymbol{v}''_{j_2} = (v''_{j_2 0}, v''_{j_2 1}, \ldots, v''_{j_2 (n-1)})$ be the vector used to calculate $h''_{j_2}(i)$. Since $j_1 < j_2$, we have $T''_{j_1}(i) \supseteq T''_{j_2}(i)$. It follows from the definition of $h''_{j_1}(i)$ and $h''_{j_2}(i)$ that

$$h''_{j_1}(i) = \min_{\boldsymbol{v}'' \in T''_{j_1}(i)} \left\{ \sum_{j \in I - \{i\}}^{n-1} \left(\phi_j - (-1)^{v''_j}\right)^2 \right\} \leq \min_{\boldsymbol{v}'' \in T''_{j_2}(i)} \left\{ \sum_{j \in I - \{i\}}^{n-1} \left(\phi_j - (-1)^{v''_j}\right)^2 \right\} = h''_{j_2}(i).$$

Furthermore, $low'_{j_1}(i, \overline{c}_i) = g(I) + h''_{j_1}(i) \leq g(I) + h''_{j_2}(i) = low'_{j_2}(i, \overline{c}_i)$.

Theorem 2

For an index $i \in I$, there exists a positive integer j such that the vector $\boldsymbol{v}''_j \in T''_j(i)$ used to calculate $h''_j(i)$ is a minimum-cost codeword whose i^{th} component is $\overline{c}_i \oplus 1$ and whose t^{th}

components are c_t^* for all $t \in \overline{I}$.

Proof of Theorem 2:

Suppose that for every positive integer j, the vector $\boldsymbol{v}_j'' = (v_{j0}'', v_{j1}'', \ldots, v_{j(n-1)}'') \in T_j''(i)$ used to calculate $h_j''(i)$ is not a codeword. Let $j_1 < j_2$ be two positive integers. Let $\boldsymbol{v}_{j_1}''$ be the vector used to calculate $h_{j_1}''(i)$ and $\boldsymbol{v}_{j_2}''$ be the vector used to calculate $h_{j_2}''(i)$. Then $T_{j_1}''(i) \supset T_{j_2}''(i)$ and $T_{j_1}''(i) \neq T_{j_2}''(i)$, which implies that $\boldsymbol{v}_{j_1}'' \neq \boldsymbol{v}_{j_2}''$. Furthermore, $T''(i)$ is a finite set of n-tuple vectors and $\boldsymbol{u}'\ \boldsymbol{G}$ is a codeword in $T''(i)$, where $\boldsymbol{u}' = (u_0', u_1', \ldots, u_{k-1}')$ is a binary k-tuple vector such that $u_i' = \overline{c}_i \oplus 1$, $u_t' = c_t^*$ for all $t \in \overline{I}$, and $u_s'' = 0$ for all $s \in \{0, 1, \ldots, k-1\} - \{i\} - \overline{I}$. It follows that there exists a positive integer j such that the vector $\boldsymbol{v}_j'' = (v_{j0}'', v_{j1}'', \ldots, v_{j(n-1)}'') \in T_j''(i)$ used to calculate $h_j''(i)$ is equal to $\boldsymbol{u}'\ \boldsymbol{G} \in T''(i)$, which is a contradiction to the assumption. Therefore, there exists some integer j such that the vector $\boldsymbol{v}_j'' \in T_j''(i)$ used to calculate $h_j''(i)$ is a codeword. Let j be the smallest positive integer such that the vector $\boldsymbol{v}_j''$ used to calculate $h_j''(i)$ is a codeword. We now prove that $\boldsymbol{v}_j''(i)$ is a minimum-cost codeword whose i^{th} component is $\overline{c}_i \oplus 1$. If $j = 1$, then according to Lemma 5, $\boldsymbol{v}_j$ is a minimum-cost codeword to the received vector in $T_j''(i)$. If $j > 1$, assume there exists another positive integer j', $0 < j' < j$ such that the vector $\boldsymbol{v}_{j'}'' = (v_{j'0}, v_{j'1}, \ldots, v_{j'(n-1)}$ $\in T_{j'}''(i)$ used to calculate $h_{j'}''(i)$ is also a codeword. According to the definition of $h_{j'}''(i)$, $T_j''(i) = T_{j'}''(i)$, which implies that $\boldsymbol{v}_j'' = \boldsymbol{v}_{j'}''$ and $h_j''(i) = h_{j'}''(i)$. It follows from Lemma 5 that $\boldsymbol{v}_j''(i)$ is a minimum-cost codeword whose i^{th} component is $\overline{c}_i \oplus 1$.

Theorem 3

For any $i \in I$, if there exists some positive integer j such that $low_j'(i, \overline{c}_i) \geq UB$, then $c_i^* = \overline{c}_i$.

Proof of Theorem 3:

For any $i \in I$, let j be the smallest positive integer such that $low_j'(i, \overline{c}_i) \geq UB$. According to Theorem 2, there exists some positive integer t such that the vector $\boldsymbol{v}_t'' \in T_t''(i)$ used to calculate $h_t''(i)$ is the minimum-cost codeword whose i^{th} component is $\overline{c}_i \oplus 1$ and whose l^{th}

component is c_l^* for every $l \in \overline{I}$. According to the definition of $h_t''(i)$, $h_{t'}''(i) = h_t''(i)$ for all positive integer $t' \geq t$. It follows from Lemma 5 that $t \geq j$ and $low_t'(i, \overline{c}_i) \geq low_j'(i, \overline{c}_i) \geq UB$. Since $low_t'(i, \overline{c}_i)$ is the cost of the minimum-cost codeword whose i^{th} component is $\overline{c}_i \oplus 1$, any codeword whose i^{th} component is $\overline{c}_i \oplus 1$ must have a cost greater than or equal to $low_t'(i, \overline{c}_i)$ (which is greater than or equal to UB). Therefore, either $\overline{c}$ is a minimum-cost codeword or the value of the i^{th} component of a minimum-cost codeword cannot be $\overline{c}_i \oplus 1$. It follows that $c_i^* = \overline{c}_i$.

References

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

[2] P. Carnevalli, L. Coletti, and S. Patarnello, "Image Processing by Simulated Annealing," *IBM J. Res. Dev.,*, vol. 29, pp. 569–579, 1985.

[3] D. Chase, "A Class of Algorithms for Decoding Block Codes with Channel Measurement Information," *IEEE Transactions on Information Theory*, pp. 170–181, January 1972.

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

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

[6] K. H. Farrell, L. D. Rudolph, C. R. P. Hartmann, and L. D. Nielsen, "Decoding by Local Optimization," *IEEE Transactions on Information Theory*, pp. 740–743, September 1983.

[7] Marc P. C. Fossorier and Shu Lin, "Soft Decision Decoding of Linear Block Codes Based on Ordered Statistics," *IEEE Transactions on Information Theory*, September 1995.

[8] Marc P. C. Fossorier and Shu Lin, "Computationally Efficient Soft Decision Decoding of Linear Block Codes Based on Ordered Statistics," *IEEE Transactions on Information Theory*, May 1996.

[9] F. Glover, "Tabu Search Methods in Artificial Intelligence and Operations Research," *ORSA Artificial Intelligence Newsletter*, vol. 1, no. 2, pp. 6, 1987.

[10] F. Glover, "Tabu Search–Part I," *ORSA Journal on Computing, Vol. 1, No. 3*, pp. 190–206, Summer 1989.

[11] Y. S. Han, C.-T. Chiu, C. R. P. Hartmann, and C. K. Mohan, "Efficient Suboptimal Decoding Linear Block Codes," (invited paper) *Proceedings of the 32nd Allerton Conference on Communication, Control, and Computing*, University of Illinois, Urbana-Champaign, September 1994.

[12] Y. S. Han and C. R. P. Hartmann, "Designing Efficient Maximum-Likelihood Soft-Decision Decoding Algorithms for Linear Block Codes Using Algorithm A^*," Technical Report SU-CIS-92-10, School of Computer and Information Science, Syracuse University, Syracuse, NY 13244, June 1992.

[13] Y. S. Han and C. R. P. Hartmann, "Efficient Optimal and Suboptimal Decoding of Linear Block Codes," *Proceedings of the 1994 SBT/IEEE International Telecommunications Symposium (ITS '94)*, Rio de Janeiro, Brazil, August 1994.

[14] Y. S. Han, C. R. P. Hartmann, and C.-C. Chen, "Efficient Priority-First Search Maximum-Likelihood Soft-Decision Decoding of Linear Block Codes," *IEEE Transactions on Information Theory*, pp. 1514–1523, September 1993.

[15] Y. S. Han, C. R. P. Hartmann, and K. G. Mehrotra, "Decoding Linear Block Codes Using a Priority-First Search: Performance Analysis and Suboptimal Version," accepted for publication in *IEEE Transactions on Information Theory.*

[16] Y. S. Han, C. R. P. Hartmann, and K. G. Mehrotra, "Efficient Suboptimal Soft-Decision Decoding of Linear Block Codes Using a Gereralization of Algorithm A^*," presented at the Recent Results Session of the *1993 IEEE International Symposium on Information Theory*, San Antonio, Texas, January 1993.

[17] Y. S. Han, C. R. P. Hartmann, and K. G. Mehrotra, "Further Results on Decoding Linear Block Codes Using a Generalized Dijkstra's Algorithm," *Proceedings of the 1994*

IEEE International Symposium on Information Theory, p. 342, Trondheim, Norway, June 1994.

[18] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transations on SSC*, vol. SSC-4, pp. 100–107, 1968.

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

[20] T. Kancko, T. Nishijima, H. Inazumi, and S. Hirasawa, "An Efficient Maximum-Likelihood-Decoding Algorithm for Linear Block Codes with Algebraic Decoder," *IEEE Transactions on Information Theory*, pp. 320–327, March 1994.

[21] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.

[22] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes.* New York, NY: Elsevier Science Publishing Company, Inc., 1977.

[23] H. Maini, K. Mehrotra, C. K. Mohan, and S. Ranka, "Genetic Algorithms for Soft-Decision Decoding of Linear Block Codes," *Journal of Evolutionary Computation*, vol. 2, pp. 145–164, January 1994.

[24] H. Maini, K. Mehrotra, C. K. Mohan, and S. Ranka, "Knowledge-Based Nonuniform Crossover," *Complex Systems*, vol. 8, pp. 257–293, 1994.

[25] K. R. Matis and J. W. Modestino, "Reduced-Search Soft-Decision Trellis Decoding of Linear Block Codes," *IEEE Transactions on Information Theory*, pp. 349–355, March 1982.

[26] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Reading, MA: Addison-Wesley Publishing Company, Inc., 1984.

[27] J. F. Shapiro, *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* New York, NY: John Wiley & Sons, Inc., 1979.

[28] C.-C. Shih, C. R. Wulff, C. R. P. Hartmann, and C. K. Mohan, "Decoding Linear Block Codes Using Optimization Techniques," *Proceedings of the 1995 IEEE International Symposium on Information Theory*, p. 414, Whistler, British Columbia, Canada, September 1995.

[29] C.-C. Shih, C. R. Wulff, C. R. P. Hartmann, and C. K. Mohan, "Heuristic Search Algorithms for Soft-Decision Decoding," (invited paper) *Proceedings of the* 33^{rd} *Allerton Conference on Communication, Control, and Computing*, pp. 690–699, University of Illinois, Urbana-Champaign, October 1995.

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

[31] D. J. Taipale and M. B. Pursley, "An Improvement to Generalized-Minimum-Distance Decoding," *IEEE Transactions on Information Theory*, pp. 167–172, January 1991.

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

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

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