Efficient Priority-First Search Maximum-Likelihood Soft-Decision Decoding of Linear Block Codes

Yunghsiang S. Han, *Student Member, IEEE,* Carlos R. P. Hartmann, *Fellow, IEEE,* and Chih-Chieh Chen, *Member, IEEE*

Abstract— **In this paper we present a novel and efficient maximum-likelihood soft-decision decoding algorithm for linear block codes. The approach used here converts the decoding problem into a search problem through a graph that is a trellis for an equivalent code of the transmitted code. A generalized Dijkstra's Algorithm, which uses a priority-first search strategy, is employed to search through this graph. This search is guided by an evaluation function f defined to take advantage of the information provided by the received vector and the inherent properties of the transmitted code. This function f is used to reduce drastically the search space and to make the decoding efforts of this decoding algorithm adaptable to the noise level. For example, for most real channels of the 35 000 samples tried, simulation results for the (128,64) binary extended BCH code show that the proposed decoding algorithm is fifteen orders of magnitude more efficient in time and in space than that proposed by Wolf. Simulation results for the (104, 52) binary extended quadratic residue code are also given.**

Index Terms— **block codes, decoding, Dijkstra's algorithm, maximum-likelihood, priority-first search, soft-decision, trellis**

I. 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 for lack of an efficient soft-decision decoding algorithm.

This paper deals with the *maximum-likelihood soft-decision 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 the 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 to hard-decision decoding.

Several researchers [6], [28], [24] have presented techniques for decoding linear block codes that convert the decoding problem into a graph-search problem on a trellis derived from the parity-check matrix of the code. Thus the MLD rule can be implemented by applying the Viterbi Algorithm [27] 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 [21].

In this paper we present a novel maximum-likelihood soft-decision decoding algorithm for linear block codes. This algorithm uses a generalization of Dijkstra's algorithm [22] to search through the trellis for a code equivalent to the transmitted code. The use of this priority-first search strategy for decoding drastically reduces the search space and results in an efficient optimal soft-decision decoding algorithm for linear block codes. Furthermore, in contrast with Wolf's algorithm [28], the decoding efforts of our decoding algorithm are adaptable to the noise level.

In Section II we review maximum-likelihood decoding of linear block codes and describe a trellis for a linear code. In Section III we give a short description of Dijkstra's Algorithm and its generalization. In Section IV we present our decoding algorithm. Simulation results for the (104,52) binary extended quadratic residue code and the (128,64) binary extended BCH code are given in Section V. Concluding remarks are presented in Section VI.

II. 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 $\boldsymbol{r} = (r_0, r_1, \ldots, r_{n-1})$, $r_j \in B$ denote the received vector, and assume that $Pr(r_j|c_i) > 0$ for $r_j \in \boldsymbol{B}$ and $c_i \in GF(2)$. Let $\hat{\boldsymbol{c}}$ be an estimate of the transmitted codeword $\boldsymbol{c}$.

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

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

Manuscript received December 21, 1991. This work was supported in part by the National Science Foundation under Grant NCR-9 205 422. The work of C. C. Chen was done while he was working under the Syracuse Center of Computational Science Research Experience for Undergraduates Program grant from the National Science Foundation NSF-REU award CDA-9 100 833.

Y. S. Han was with the School of Computer and Information Science at Syracuse University, Syracuse, NY 13244. He is now with the Department of Electronic Engineering, Hua Fan Institute of Humanities and Technology, Taipei Hsien, Taiwan, R.O.C.

C. R. P. Hartmann is with the School of Computer and Information Science at Syracuse University, Syracuse, NY 13244.

C. C. Chen was with the Department of Electrical and Computer Engineering, The Johns Hopkins University, Baltimore, MD 21218. He is now with the Department of Computer Science, University of California, Los Angeles, CA 90024.

IEEE Log Number 9211655.

Following the formulation given in [17], we define the bit log-likelihood ratio of r_j as

$$\phi_j = \ln \frac{Pr(r_j|0)}{Pr(r_j|1)}.$$

Furthermore, let $\boldsymbol{\phi} = (\phi_0, \phi_1, \ldots, \phi_{n-1})$. By [17, Theorem 5] the MLD rule can be written as

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

$$\sum_{j=0}^{n-1} (\phi_j - (-1)^{c_{\ell j}})^2 \le \sum_{j=0}^{n-1} (\phi_j - (-1)^{c_{ij}})^2 \quad \text{for all } \boldsymbol{c}_i \in \boldsymbol{C}. \tag{1}$$

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

We now give a short description of a trellis [1] for the code $\boldsymbol{C}$ where the search will be performed. Let $\boldsymbol{H}$ be a parity-check matrix of $\boldsymbol{C}$, and let $\boldsymbol{h}_i$, $0 \le i < n$ be the column vectors of $\boldsymbol{H}$. Furthermore, let $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$ be a codeword of $\boldsymbol{C}$. With respect to this codeword, we recursively define the states $\boldsymbol{s}_t$, $-1 \le t < n$, as

$$s_{-1} = \mathbf{0}$$

and

$$s_t = s_{t-1} + c_t h_t = \sum_{i=0}^{t} c_i h_i, \quad 0 \le t < n.$$

Clearly, $\boldsymbol{s}_{n-1} = \mathbf{o}$ for all codewords of $\boldsymbol{C}$. The above recursive equation can be used to draw a trellis diagram. In this trellis, $\boldsymbol{s}_{-1} = \mathbf{0}$ identifies the start node that is at level -1; $\boldsymbol{s}_{n-1} = \mathbf{0}$ identifies the goal node that is at level $n-1$; and each state $\boldsymbol{s}_t, 0 \le t < n-1$ identifies a node at level t. Furthermore, each transition (arc) is labeled with the appropriate codeword bit c_t. Thus, there is a one-to-one correspondence between the codewords of $\boldsymbol{C}$ and the sequences of labels encountered when traversing a path in the trellis from start node to the goal node. A more detailed description of a trellis for a linear block code can be found in [28]. Note that the trellis defined here corresponds to the expurgated trellis of [28].

In order to implement the MLD rule using the trellis for $\boldsymbol{C}$, we need to associate a cost to every arc in this trellis. Therefore, the cost of the arc from $\boldsymbol{s}_{t-1}$ to $\boldsymbol{s}_t = \boldsymbol{s}_{t-1} + c_t \boldsymbol{h}_t$ is assigned the value $(\phi_t - (-1)^{c_t})^2$. The solution of the decoding problem is thus converted to finding a path from the start node to the goal node, that is, a codeword $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$ such that $\sum_{i=0}^{n-1} (\phi_i - (-1)^{c_i})^2$ is minimum among all paths from the start node to the goal node. Such a path is denoted as an optimal path.

Wolf's algorithm [28] finds an optimal path by applying the Viterbi algorithm [27] to search through the trellis for $\boldsymbol{C}$. Thus, this decoding algorithm uses a breadth-first search strategy to accomplish this search. The time and space complexities of Wolf's algorithm are of $O(n \times \min(2^k, 2^{n-k}))$ [10], since it traverses the entire trellis. However, Forney has given a procedure to reduce the number of states in the trellis for $\boldsymbol{C}$ [14].

In order to avoid traversing the entire trellis, a different search strategy—such as the priority-first search—must be employed. In the next section we present a generalization of the well-known Dijkstra's algorithm, which uses such a strategy [22].

III. Generalized Dijkstra's Algorithm

Dijkstra's Algorithm (DA) [11] is usually employed to find a minimum cost path from the start node to every other node of a weighted directed graph (in our case, a trellis). In this case the algorithm visits all the nodes in the graph, but can easily be modified to find a minimum cost path from the start node to the goal node, that is, an optimal path. This modified algorithm may not visit all the nodes in the graph. We now give a short description of this algorithm.

As in Wolf's algorithm, a node in the trellis may be visited twice. Thus, for each visited node m, we store the path $\boldsymbol{P'}_m$ from the start node to node m with minimum cost $g(m)$ found so far by the algorithm. That is, if $\boldsymbol{P'}_m$ has labels $\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell$, then $g(m) = \sum_{i=0}^{\ell} (\phi_i - (-1)^{\overline{v}_i})^2$. Note that the value of $g(m)$ may be updated, since the minimum cost path from start node to node m may change as the search progresses.

It will be convenient to introduce the notion of a *successor operator* that is applied to a node. This operator, when applied to a node m, (1) gives all the immediate successors of node m; (2) for every immediate successor of node m, checks if such a node was visited before; (3) for every immediate successor of node m, stores the minimum cost path from the start node to this node and the cost of this path found so far by the algorithm. We call this process of applying the successor operator to a node *expanding* the node.

DA maintains two lists of nodes of the given trellis, namely, list CLOSED and list OPEN. List CLOSED contains the set of nodes that were expanded. List OPEN contains the set of nodes that were visited, but not expanded. The algorithm selects node m on list OPEN with minimum $g(m)$. It expands this node and inserts it into list CLOSED. It inserts into list OPEN only the immediate successors of node m that have not been expanded before. When the algorithm selects to expand the goal node it has found an optimal path.

The correctness of this algorithm is based on the following fact: When a node is selected for expansion, the algorithm has already found a minimum cost path from the start node to this node [11] and, consequently, we do not need to update the lowest cost path from the start node to any node that is already on list CLOSED. Furthermore, when expanding node m, we do not need to update the lowest cost path from the start node to any descendant of an immediate successor of node m that is already on list CLOSED.

The problem of determining whether a newly visited node is on list OPEN or list CLOSED can be computationally expensive, and we may therefore decide to avoid making this check, in which case the search tree may contain several repeated

nodes, and list CLOSED does not need to be maintained. These node repetitions lead to redundant successor computations and there is a trade-off between the computation cost of testing for repeated nodes and the computation cost of generating a larger search tree. Note that in DA, testing for repeated nodes is performed.

As pointed out before, DA associates to every node m visited by the value $g(m)$. This value can be treated as an estimate of the cost of the minimum cost path from the start node to node m. However, in many graph search problems we may be able to obtain an estimate, $h(m)$, of the cost of the minimum cost path from node m to the goal node. Thus $f(m) = g(m) + h(m)$ can be treated as an estimate of the minimum cost path from the start node to the goal node that goes through node m. We will impose $h(\boldsymbol{s}_{n-1}) = 0$ on the heuristic function since $\boldsymbol{s}_{n-1}$ is the goal node and no estimation is necessary for it. The Generalized Dijkstra's Algorithm (GDA) uses $f(m)$ instead of $g(m)$ to guide the search through the trellis. Function h is known as a heuristic function, and function f as an evaluation function [22].

In order to guarantee that GDA finds an optimal path, we impose the following condition on the heuristic function h.

Condition: For all nodes m_i and m_j such that node m_j is an immediate successor of node m_i,

$$h(m_i) \leq h(m_j) + c(m_i, m_j), \tag{2}$$

where $c(m_i, m_j)$ is the arc cost between node m_i and node m_j.

The condition guarantees that when a node is selected for expansion GDA has already found a minimum cost path from the start node to this node. Thus when GDA selects to expand the goal node, it has already found an optimal path. The proof that GDA finds an optimal path is given in Appendix A.

From now on we will assume that the function h used in GDA satisfies the condition.

The following results of GDA will be used in the design of our decoding algorithm. Proofs of these results can be found in Appendix B.

Result 1) For every node m,

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

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

Result 2) If node m_i is selected for expansion, then $f(m_i) \leq f(m_j)$, where m_j is an immediate successor of node m_i, which is not on list CLOSED.

Result 3) Let $\boldsymbol{P}$ be a path found by GDA from the start node to the goal node with cost UB. GDA still finds an optimal path if it removes from list OPEN any node m for which $f(m) \geq UB$.

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

Finally, we remark that GDA is a particular case of Algorithm A^* that is widely used in Artificial Intelligence search problems [22]. Algorithm A^* finds an optimal path, but it imposes less restriction on the heuristic function h than we have imposed. Furthermore, Algorithm A^* can be considered as a branch-and-bound type algorithm. In general, it is difficult to give any idea of how well a branch-and-bound algorithm will perform on a given problem. Nevertheless, the technique is sufficiently powerful that it is often used in practical applications [7].

IV. Decoding Algorithm

Our decoding algorithm uses GDA to search through a trellis for a code $\boldsymbol{C}^*$ that is equivalent to code $\boldsymbol{C}$. $\boldsymbol{C}^*$ is obtained from $\boldsymbol{C}$ by permuting the positions of codewords of $\boldsymbol{C}$ in such a way that the first k positions of codewords in $\boldsymbol{C}^*$ correspond to the "most reliable linearly independent" positions in the vector $\boldsymbol{\phi}$. Let $\boldsymbol{G}^*$ be a generator matrix of $\boldsymbol{C}^*$ whose first k columns form the $k \times k$ identity matrix. The time complexity of the procedure to construct $\boldsymbol{G}^*$ is $O(k^2 \times n)$; however, many of the operations performed during this construction can be done in parallel. In this case, the time complexity becomes $O(k \times n)$.

In our decoding algorithm the vector $\boldsymbol{\phi}^* = (\phi_0^*, \phi_1^*, \ldots, \phi_{n-1}^*)$ is used as the "received vector." It is obtained by permuting the positions of $\boldsymbol{\phi}$ in the same manner in which the columns of $\boldsymbol{G}$ are permuted to obtain $\boldsymbol{G}^*$.

GDA, guided by an evaluation function f, searches through a trellis for $\boldsymbol{C}^*$. As noted before, function f is defined for every node m in the trellis as

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

where $g(m)$ is the lowest cost path from the start node to node m found so far by the algorithm.

We now define our heuristic function h, which satisfies the condition. In order to define a function h that is a "good" estimator of h^* we must use properties of the linear block code that are invariant under any permutation of the positions of the codewords.

Let $HW = \{w_i | 0 \leq i \leq I\}$ be the set of all distinct Hamming weights that codewords of $\boldsymbol{C}$ may have. Furthermore, assume $w_0 < w_1 < \cdots < w_I$. Our heuristic function is defined to take into consideration the linear property of $\boldsymbol{C}^*$ and that the Hamming distance between any two codewords of $\boldsymbol{C}^*$ must belong to HW.

Let $\boldsymbol{c}^*$ be a given codeword of $\boldsymbol{C}^*$. Our function h will be defined with respect to $\boldsymbol{c}^*$, which is called the seed of the decoding algorithm.

1) For nodes at level $\ell, -1 \leq \ell < k-1$:
Let m be a node at level ℓ, and let $\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell$ be the labels of the lowest cost path $\boldsymbol{P'}_m$ from the start node to node m found so far by the algorithm. We now construct the set, $T(m)$, of all binary n-tuples $\boldsymbol{v}$ such that their first $\ell + 1$ entries are the labels of $\boldsymbol{P'}_m$ and $d_H(\boldsymbol{v}, \boldsymbol{c}^*) \in HW$, where $d_H(\boldsymbol{x}, \boldsymbol{y})$ is the Hamming distance between $\boldsymbol{x}$ and $\boldsymbol{y}$. That is,

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

Note that $T(m) \neq \emptyset$. This can easily be seen by considering the binary k-tuple $\boldsymbol{u} = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, 0, \ldots, 0)$ and noting that $\boldsymbol{u} \cdot \boldsymbol{G}^* \in T(m)$.

We now define function h as

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

2) For nodes at level $\ell, k-1 \le \ell < n$: Because of the linear property of $\boldsymbol{C}^*$ and the fact that the first k columns of $\boldsymbol{G}^*$ are linearly independent, there is only one path from any node at level $k-1$ to the goal node. Furthermore, we can easily determine the labels $v_k^*, v_{k+1}^*, \ldots, v_{n-1}^*$ of this path by using $\boldsymbol{G}^*$ and calculate its cost as $\sum_{i=k}^{n-1} \left(\phi_i^* - (-1)^{v_i^*}\right)^2$. In view of the above fact, we define function h as

$$h(m) = \sum_{i=\ell+1}^{n-1} \left(\phi_i^* - (-1)^{v_i^*}\right)^2,$$

where $v_{\ell+1}^*, v_{\ell+2}^*, \ldots, v_{n-1}^*$ are the labels of the only path $\boldsymbol{P}_m$ from node m to the goal node. Note that if node m is the goal node, then $h(m) = 0$. Furthermore, $h(m) = h^*(m)$ since there is only one path from node m to the goal node and $h(m)$ is the cost of this path.

The proof that our heuristic function satisfies the condition is given in Appendix C.

For a given seed $\boldsymbol{c}^*$, the evaluation function f is $f(m) = g(m) + h(m)$. It is very important that the time complexity for calculating $h(m)$ be "reasonable," for otherwise the time taken by the decoding algorithm is spent calculating $h(m)$, even though there are only a few nodes to be visited in the trellis. In Appendix D we present an algorithm, whose time complexity is $O(n)$, to calculate $h(m)$ for node m at level $\ell, -1 \le \ell < k-1$.

We now give a property of our heuristic function h that will be used to speed up the decoding procedure. The proof of this property is given in Appendix E.

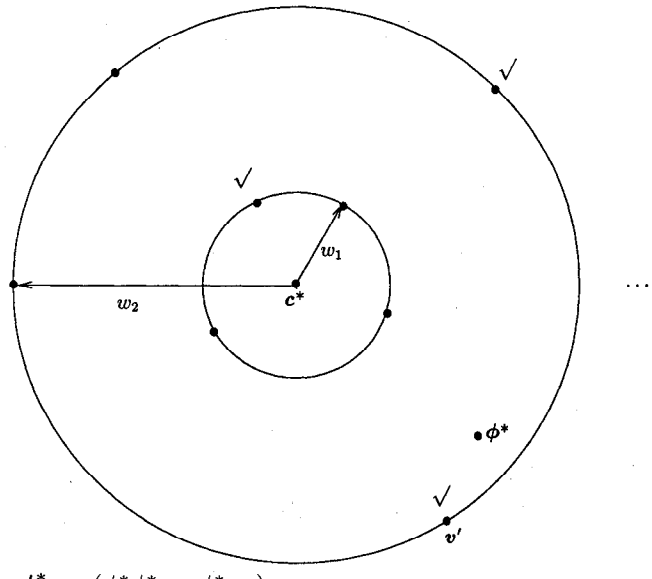
Property: For a given seed $\boldsymbol{c}^*$, if nodes $m_{\ell 1}$ and $m_{\ell 2}$ are immediate successors of node m_j, then

$$f(m_{\ell 1}) = f(m_j) \text{ or } f(m_{\ell 2}) = f(m_j), \text{ where } -1 \le j < k-2.$$

We now give a geometric interpretation of $f(m)$ for a node m at level $\ell, -1 \le \ell < k-1$. Consider the set of all n-tuples over the real numbers. Fig. 1 depicts $\boldsymbol{c}^*$, $\boldsymbol{\phi}^*$, and all the points $\boldsymbol{v} = (v_0, v_1, \ldots, v_{n-1})$ whose entries are 0 and 1 and $d_H(\boldsymbol{c}^*, \boldsymbol{v}) \in HW$. For calculating $f(m)$, we consider the lowest cost path $\boldsymbol{P'}_m$ from the start node to node m found so far by the algorithm. Let $\overline{v}_0, \overline{v} \ldots, \overline{v}_\ell$ be the labels of $\boldsymbol{P'}_m$. Now we consider only those points $\boldsymbol{v}$ defined above whose first $\ell+1$ entries are $\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell$. In Fig. 1 they are indicated by a check mark. From the points with check marks we select one that minimizes $\sum_{i=0}^{\ell} \left(\phi_i^* - (-1)^{\overline{v}_i}\right)^2 + \sum_{i=\ell+1}^{n-1} (\phi_i^* - (-1)^{v_i})^2$. This point is $\boldsymbol{v'} = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, v'_{\ell+1}, \ldots, v'_{n-1})$ in Fig. 1. Thus

$$f(m) = \sum_{i=0}^{\ell} \left(\phi_i^* - (-1)^{\overline{v}_i}\right)^2 + \sum_{i=\ell+1}^{n-1} \left(\phi_i^* - (-1)^{v'_i}\right)^2.$$

It may be impossible to determine the set HW for some block codes; however, our algorithm will still find an optimal path even if in the computation of function h the algorithm considers all the Hamming weights of any superset of HW. If the optimal path is unique, then the decoding algorithm using HW will never expand more nodes than an algorithm using a superset of HW. This result is a consequence of the following property of GDA: If there are two evaluation functions $f_1(m) = g_1(m) + h_1(m)$ and $f_2(m) = g_2(m) + h_2(m)$ satisfying $h_1(m) < h_2(m)$ for every node m, the GDA using evaluation function f_2 will never expand more nodes than the GDA using evaluation function f_1 [22]. Furthermore, if there exists a unique optimal path, then the above results hold when $h_1(m) \le h_2(m)$ is satisfied for every node m.



$\boldsymbol{\phi}^* = (\phi_0^* \phi_1^* \cdots \phi_{n-1}^*)$
$\boldsymbol{v'} = (\overline{v}_0, \overline{v}_1, \cdots, \overline{v}_l, v'_{l+1}, \cdots, v'_{n-1})$
$f(m) = \sum_{i=0}^{l} (\phi_i^* - (-1)^{\overline{v}_i})^2 + \sum_{i=l+1}^{n-1} (\phi_i^* - (-1)^{v'_i})^2$

Fig. 1. Geometric interpretation of $f(m)$

We now present some properties of the decoding algorithm that can be used to speed up the decoding procedure. By the property, when our algorithm expands a node m at level $\ell < k-2$, we need to compute the value of function f for only one of its successors, because the value of function f for the other successor is equal to that of node m and we can easily determine which successor has the value $f(m)$. Thus our algorithm is a depth-first search type algorithm.

Let node m_1 at level $\ell < k-2$ be the node on list OPEN selected for expansion. Let $h(m_1) = \sum_{i=\ell+1}^{n-1} (\phi_i^* - (-1)^{v_i})^2$. Consider now the path $\boldsymbol{P}_{m_1,m_2}$ from node m_1 to node m_2 at level $k-2$, whose labels are $v_{\ell+1}, v_{\ell+2}, \ldots, v_{k-2}$. It is easily seen by the property and the definition of our function h that the value of function f for every node in path $\boldsymbol{P}_{m_1,m_2}$ is equal to $f(m_1)$. Furthermore, by Result 2 we can conclude that path $\boldsymbol{P}_{m_1,m_2}$ will be the path followed by the algorithm. Thus, we do not have to calculate the values of function f for the nodes of this path, which reduces considerably the time complexity of the algorithm.

Our algorithm will search the trellis only up to level $k-1$, since we can construct the only path from any node m at level

$k-1$ to the goal node using $\boldsymbol{G}^*$. The labels of the combined paths from the start node to node m, and from node m to the goal node, correspond to a codeword. So the cost of this path, which is equal to $f(m)$, can be used as an upper bound on the cost of an optimal path. By Result 3, we can use this upper bound to reduce the size of list OPEN.

The trellis search can be stopped at any time when we know that a generated codeword $\boldsymbol{c}_\ell^* = \left(c_{\ell 0}^*, c_{\ell 1}^*, \cdots, c_{\ell(n-1)}^*\right)$ satisfies Inequality 1. The following criterion can be used to indicate this fact.

Criterion: If $h(\boldsymbol{s}_{-1}) = \sum_{j=0}^{n-1} \left(\phi_j^* - (-1)^{c_{\ell j}^*}\right)^2$, where $h(\boldsymbol{s}_{-1})$ is calculated with respect to seed c_ℓ^*, then c_ℓ^* satisfies Inequality 1.

Recall that $\boldsymbol{s}_{-1}$ is the start node.

The validity of the criterion is based on the fact that, since $\boldsymbol{C}^* \subseteq T(\boldsymbol{s}_{-1})$, then $h(mbis_{-1}) \leq \sum_{j=0}^{n-1} \left(\phi_j^* - (-1)^{c_{tj}^*}\right)^2$ for any $mbic_t^* \in mbiC^*$. Note that the decision criterion introduced in [25] is equivalent to the criterion.

It is important to mention that seed $\boldsymbol{c}^*$ does not need to be fixed during the decoding of $\boldsymbol{\phi}^*$. When seed $\boldsymbol{c}^*$ is allowed to change, we have an adaptive decoding procedure. In order to avoid increasing computation time when the seed is changed at some stage of the decoding procedure, we may not want to recalculate the values of function h with respect to this new seed for every node on list OPEN. Under these circumstances, nodes on list OPEN may have values of function h calculated with respect to different seeds; thus we can no longer guarantee that Inequality 2 will be satisfied, and we cannot assure that when a node is selected for expansion the decoding algorithm has already found a minimum cost path from the start node to this node. Therefore, the decoding algorithm may not find an optimal path, but by not checking for repeated nodes it is ensured that the decoding algorithm will find an optimal path. This can easily be seen, since the procedure will now generate a decision tree, and $h(m) \leq h^*(m)$ for every node of the tree, independently of the seed used to compute $h(m)$. Thus, $f(m) \leq g(m) + h^*(m)$. As the procedure is now generating a decision tree, the cost of the minimum cost path from the start node to the goal node that goes through node m is $g(m)+h^*(m)$. If we do not check for repeated nodes, then the adaptive version of GDA will never delete all optimal paths during the search procedure.

V. Simulation Results For The AWGN Channel

In this section we present simulation results for the $(104, 52)$ binary extended quadratic residue code and the $(128, 64)$ binary extended BCH code when these codes are transmitted over the additive white Gaussian noise (AWGN) channel. We assume that antipodal signaling is used in the transmission so that the j^{th} components of the transmitted codeword $\boldsymbol{c}$ and received vector $\boldsymbol{r}$ are

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

respectively, where E is the signal energy per channel bit and e_j is a noise sample of a Gaussian process with single-sided noise power per hertz N_0. The variance of e_j is $N_0/2$ and the signal to noise ratio (SNR) for the channel is $\gamma = E/N_0$. In order to account for the redundancy in codes of different rates, we used the SNR per transmitted information bit $\gamma_b = E_b/N_0 = \gamma n/k$ in our simulation. For the AWGN channel, $|mbi\phi = \frac{4\sqrt{E}}{N_0}\boldsymbol{r}$ [17], so we can substitute $\boldsymbol{r}(\boldsymbol{r}^*)$ for $\boldsymbol{\phi}(\boldsymbol{\phi}^*)$ in our decoding algorithm.

We do not know HW for these two codes, so we use a superset for them. For (104,52) we know that $d_{\min} = 20$ and that the Hamming weight of any codeword is divisible by 4 [19]. Thus for this code the superset used is $\{x|(x$ is divisible by 4 and $20 \leq x \leq 84)$ or $(x = 0)$ or $(x = 104)\}$; for (128,64), the superset used is $\{x|$ (x is even and $22 \leq x \leq 106)$ or $(x = 0)$ or $(x = 128)\}$, since this code has $d_{\min} = 22$.

In the implementation of our decoding algorithm we decided not to check for repeated nodes. In this situation the graph becomes a decision tree, thus we need not keep list CLOSED. Furthermore, list OPEN is always kept ordered according to the values f of its nodes.

If we assume that, 1) the time complexity of the algorithm that constructs $\boldsymbol{G}^*$ is $O(k \times n)$ (implemented by performing operations in parallel), 2) the data structure used to implement list OPEN is a B-tree [26], and 3) the time complexity of the algorithm that constructs the codeword $\boldsymbol{u} \cdot \boldsymbol{G}^*$ is $O(n)$ (implemented by performing the operation in parallel), then the time complexity and the space complexity of our algorithm are $O(n \times N(\boldsymbol{r}))$ and $O(n \times M(\boldsymbol{r}))$, respectively, where

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

The values of $N(\boldsymbol{r})$ and $M(\boldsymbol{r})$ will strongly depend upon SNR. Up to now we do not have a "good" estimator of these values; however, they are upperbounded by $2^{k+1}-1$. In the worst case, therefore, the time and space complexities of our algorithm are $O(n \times 2^k)$, which are, under the condition $k \leq (n-k)$, equal to those of Wolf's algorithm [28], which are $O(n \times \min(2^k, 2^{n-k}))$ [10].

Since our decoding algorithm will generate a decision tree, we have implemented its adaptive version without compromising its optimality. The seed $\boldsymbol{c}^*$ is updated according to the following rule: For every codeword $\boldsymbol{c}_1^*$ generated during the decoding of $\boldsymbol{r}^*$, if the value of $h(\boldsymbol{s}_{-1})$ calculated with respect to $\boldsymbol{c}_1^*$ is greater than the value of $h(\boldsymbol{s}_{-1})$ calculated with respect to $\boldsymbol{c}^*$, then set $\boldsymbol{c}_1^*$ as the seed. The rationale behind this rule is that, for any node m, $h(m) \geq h(\boldsymbol{s}_{-1})$ whenever these values are calculated with respect to the same seed. We remark here that we did not recalculate the values of function f with respect to the new seed for the nodes on list OPEN. Simulation results attested to the fact that the efficiency of this decoding algorithm depends strongly on the selection of the initial seed.

TABLE I
SIMULATION FOR THE $(104, 52)$ CODE

γ_b	5 dB		6 dB		7 dB		8 dB	
	max	ave	max	ave	max	ave	max	ave
$N(\boldsymbol{r})$	142123	19	2918	1	221	1	0	0
$C(\boldsymbol{r})$	32823	5	519	2	35	2	1	1
$M(\boldsymbol{r})$	13122	4	1912	1	155	1	0	0

TABLE II
BIT ERROR PROBABILITY AND CODING GAIN FOR THE (104,52) CODE

γ_b	5 dB	6 dB	7 dB	8 dB
P_b	$2.028 \times 10^{-10^*}$	$5.023 \times 10^{-14^*}$	$1.494 \times 10^{-18^*}$	$3.079 \times 10^{-24^*}$
CG	7.90	8.35	8.80	9.05

In our implementation the initial seed $\boldsymbol{c}_0^*$ is obtained as follows. Let $\boldsymbol{u} = (u_0, u_1, \ldots, u_{k-1})$ where

$$u_i = \begin{cases} 0 & \text{if } r_i^* \geq 0; \\ 1 & \text{if } r_i^* < 0; \end{cases},$$

and $\boldsymbol{r}^* = (r_0^*, r_1^*, \ldots, r_{k-1}^*, r_k^*, \ldots, r_{n-1}^*)$. Now, $\boldsymbol{c}_0^* = \boldsymbol{u} \cdot \boldsymbol{G}^*$.

First, we give simulation results for the (104,52) binary extended quadratic residue code. Quadratic residue codes are known to be very good codes that are difficult to decode even when only hard-decision decoding is employed [4], [8], [5], [23]. Some quadratic residue codes have been decoded by using information-set decoding algorithms [3]. However, these algorithms are suboptimal, that is, they do not implement the MLD rule. The only two maximum-likelihood soft-decision decoding algorithms known to us that can be used to decode the (104,52) code are Wolf's algorithm [28] and Hwang's algorithm [17].

It is difficult to compare the performance of our algorithm with that of Hwang's, because he found the subset of codewords that must be stored for implementing the MLD rule only for very short codes [17, Table I]. However, we observe that the complexities of Wolf's algorithm are approximately the same as those of Hwang's for the codes presented in Table I of [17]. More evidence of this claim can be obtained by using the results presented in [9]. We will therefore compare the performance of our algorithm to that of Wolf's. We will assume for comparison purposes that the time and space complexities of Wolf's algorithm are of $O(n \times \min(2^k, 2^{n-k}))$, since it is difficult to find, using Forney's procedure [14], a trellis with minimum number of states for the $(104, 52)$ code.

Simulation results for the $(104, 52)$ code for γ_b equal to 5 dB, 6 dB, 7 dB, and 8 dB are given in Table I. These results were obtained by simulating 35 000 samples for each SNR. Note that the time and space complexities of Wolf's algorithm are proportional to $2^{52} \approx 4.50 \times 10^{15}$, where

1) $N(\boldsymbol{r})$ = the number of nodes visited during the decoding of $\boldsymbol{r}$;
2) $C(\boldsymbol{r})$ = number of codewords constructed in order to decide on the closest codeword to $\boldsymbol{r}$;
3) $M(\boldsymbol{r})$ = maximum number of nodes stored on list OPEN during the decoding of $\boldsymbol{r}$;
4) max = maximum value among 35 000 samples;
5) ave = average value among 35 000 samples;
6) $\gamma_b = E_b/N_0$.

TABLE III
DISTRIBUTION OF $N(\boldsymbol{r})$, $C(\boldsymbol{r})$, AND $M(\boldsymbol{r})$ FOR THE (104,52) CODE FOR $\gamma_b = 5$ dB

Interval	Frequencies		
	$N(\boldsymbol{r})$	$C(\boldsymbol{r})$	$M(\boldsymbol{r})$
0	34030	0	34196
1–2000	953	34994	801
2001–4000	8	2	0
4001–6000	1	1	0
6001–8000	1	0	0
8001–10 000	1	0	1
10 001–18 000	2	0	2
18 001–40 000	1	3	0
40 001–144 000	3	0	0
more than 144 000	0	0	0

TABLE IV
SIMULATION FOR THE $(128, 64)$ CODE

γ_b	5 dB		6 dB		7 dB		8 dB	
	max	ave	max	ave	max	ave	max	ave
$N(\boldsymbol{r})$	216052	42	13603	2	1143	1	0	0
$C(\boldsymbol{r})$	38219	8	1817	2	91	2	1	1
$M(\boldsymbol{r})$	16626	7	856	1	965	1	0	0

Since during simulation no decoding errors occurred for any of the above SNR's, the bit error probability is estimated using the formula [13]

$$n_d \sqrt{d_{\min}/(4\pi n k \gamma_b)}\; e^{-(k d_{\min} \gamma_b / n)}, \tag{3}$$

where n_d is the number of codewords of Hamming weight $d_{\min}$. The value of n_d was calculated using the results presented in [20]. Table II gives an estimate of the bit error probability and coding gain for above SNR's.

1) P_b = bit error probability;
2) CG = coding gain (dB);
3) * Calculate using (3).

The distributions of $N(\boldsymbol{r})$, $C(\boldsymbol{r})$, and $M(\boldsymbol{r})$ for the (104,52) code for γ_b equal to 5 dB are given in Table III.

We now give the simulation results for the (128,64) code. Since an algebraic decoder that corrects up to 10 bit errors can be constructed for this code, the maximum-likelihood soft-decision decoding algorithm recently proposed in [18] can be implemented. However, in this paper simulation results are given only for very short codes up to length 23. Suboptimal decoding procedures for this code have been proposed in [12], [3]. Again, we will assume for comparison purposes that the time and space complexities of Wolf's algorithm are of $O(n \times \min(2^k, 2^{n-k}))$, since it is very difficult to find, using Forney's procedure [14], a trellis with the minimum number of states for the $(128, 64)$ code. Note that the time and space complexities of Wolf's algorithm are proportional to $2^{64} \approx 1.84 \times 10^{19}$.

Simulation results for the (128,64) code for γ_b equal to 5 dB, 6 dB, 7 dB, and 8 dB are given in Table IV. These results were obtained by simulating 35 000 samples for each SNR.

TABLE V
BIT ERROR PROBABILITY AND CODING GAIN FOR THE (128,64) CODE

γ_b	5 dB	6 dB	7 dB	8 dB
P_b	1.57×10^{-12}*	1.71×10^{-16}*	1.82×10^{-21}*	1.02×10^{-27}*
CG	8.85	9.22	9.50	9.70

Table V gives only an estimate of the bit error probability and coding gain for above SNR's, because no decoding error occurred during simulation.

When calculating P_b using (3), the value of $n_d = 243840$ was taken from [2].

The distributions of $N(\boldsymbol{r})$, $C(\boldsymbol{r})$, and $M(\boldsymbol{r})$ for the (128,64) code for γ_b equal to 5 dB are given in Table VI.

Simulation results for these codes show that for the 35 000 samples tried, a drastic reduction on the search space was achieved for most practical communication systems where the probability of error is less than 10^{-3} (γ_b greater than 6.8 dB) [8], even when the algorithm uses a superset of HW.

In order to verify the contribution of our heuristic function h to the efficiency of our decoding algorithm, we implemented DA with our speed-up techniques for the $(128, 64)$ code. Simulation results for 6 dB indicate that for the two samples that did not satisfy the criterion among the 35 000 samples, more than 350 000 nodes needed to be stored. On the other hand, our algorithm needed to store at most 856 nodes to decode these samples.

Simulation results showed that our adaptive decoding algorithm described in this section is at least one order of magnitude more efficient in time and space than that proposed in [15], where the seed is $\boldsymbol{c}^* = \mathbf{0}$ during the entire decoding procedure.

VI. CONCLUSION

In this paper we have proposed a novel decoding technique. Simulation results for the above linear block codes show that for the 35 000 samples tried this decoding technique drastically reduced the search space, especially for most practical communication systems where the probability of error is less than 10^{-3} (γ_b greater than 6.8 dB) [8]. For example, the results of Table 4 at 6 dB show that, for the 35 000 samples tried, in the worst case this decoding algorithm is approximately 15 orders of magnitude more efficient in time and space than Wolf's.

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 a heuristic function that takes advantage of the specific properties of this code; (4) any stopping criterion can be easily incorporated into it.

Furthermore, we would like to point out that the algorithm present in this paper is suitable for a parallel implementation, one reason being that when calculating $h(m)$ for node m, the algorithm has determined the labels of the path from node m to a node at level $k-2$ that it will follow, so the successors of the nodes in this path can be open simultaneously and processed independently. This will 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 algorithm.

TABLE VI
DISTRIBUTION OF $N(\boldsymbol{r})$, $C(\boldsymbol{r})$, AND $M(\boldsymbol{r})$ FOR THE (128,64) CODE FOR $\gamma_b = 5$ dB

Interval	Frequencies		
	$N(\boldsymbol{r})$	$C(\boldsymbol{r})$	$M(\boldsymbol{r})$
0	33614	0	33893
1–2000	1324	34988	1096
2001–4000	21	2	4
4001–6000	8	2	4
6001–8000	7	0	0
8001–10 000	7	4	0
10 001–18 000	7	2	3
18 001–40 000	4	2	0
40 001–218 000	8	0	0
more than 218 000	0	0	0

We remark here that for computing the heuristic function h we used only one seed. However, we can generalize the procedure to calculate function h with respect to several seeds. Details of this approach can be found in [16].

We conjecture that this decoding algorithm will be efficient for most practical communication systems where the probability of error is less than 10^{-3} (γ_b greater than 6.8 dB). Since the number of nodes opened during the decoding procedure is a random variable, in order to verify this conjecture the probability distribution of the number of nodes visited must be computed in order to determine the computational performance of the proposed decoding algorithm.

For low SNR and for codes of moderate to long lengths, the number of nodes opened during the decoding procedure may be great and the proposed decoding algorithm impractical. Thus the development of a suboptimal decoding algorithm based on GDA seems to be a promising line of research.

APPENDIX A

In this appendix we prove by contradiction that if the condition is satisfied, then when a node is selected for expansion, GDA has already found a minimum cost path from the start node to this node. Thus, when the algorithm selects to expand the goal node, it has found an optimal path.

Let node m_t be any node selected for expansion by GDA. Assume that the path from the start node to node m_t found so far by GDA is not a minimum cost path from the start node to node m_t. Let $\mathrm{P}'^{*}_{m_t} = (m_{-1}, m_0, \ldots, m_\ell, \ldots, m_{t-1}, m_t)$ be a minimum cost path from the start node to node m_t. Let node m_ℓ be the first node in this sequence of nodes that is on list OPEN. Furthermore, let $g^*(m)$ be the actual cost of a minimum cost path from the start node to node m. By the condition,

$$g^*(m_{t-1}) + h(m_{t-1}) \le g^*(m_{t-1}) + h(m_t) + c(m_{t-1}, m_t) = g^*(m_t) + h(m_t).$$

By transitivity we have that

$$g^*(m_\ell) + h(m_\ell) \le g^*(m_t) + h(m_t).$$

Since $g^*(m_\ell) + h(m_\ell) = f(m_\ell)$ and $g^*(m_t) < g(m_t)$, then $f(m_\ell) < f(m_t)$. Contradiction. □

APPENDIX B

Proof of Result 1.

Let m_t be any node in the trellis. Let $\boldsymbol{P}^*_{m_t} = (m_t, m_{t+1}, \ldots, m_{n-2}, m_{n-1})$ be a minimum cost path from m_t to the goal node. Thus, the cost of this path is $h^*(m_t) = \sum_{i=t}^{n-2} c(m_i, m_{i+1})$. By the condition, $h(m_{n-2}) \leq h(m_{n-1}) + c(m_{n-2}, m_{n-1})$. By transitivity, $h(m_t) \leq h(m_{n-1}) + \sum_{i=t}^{n-2} c(m_i, m_{i+1})$. Since $h(m_{n-1}) = 0$, then $h(m_t) \leq h^*(m_t)$. □

Proof of Result 2.

Assume node m_i is selected for expansion. We now consider two cases:

Case 1. m_j was not visited. In this case

$$f(m_j) = g^*(m_i) + c(m_i, m_j) + h(m_j),$$

where $g^*(m)$ is the actual cost of a minimum cost path from the start node to node m. By the condition, $f(m_j) \geq g^*(m_i) + h(m_i) = f(m_i)$.

Case 2. m_j is on list OPEN. In this case, since m_j is not selected for expansion or by Case 1, then $f(m_j) \geq f(m_i)$. □

Proof of Result 3.

Consider an optimal path $\boldsymbol{P}^* = (m_{-1}, m_0, \ldots, m_\ell, \ldots, m_{n-1})$. Let node m_ℓ be the first node in this sequence of nodes that is on list OPEN. Thus

$$f(m_\ell) = g^*(m_\ell) + h(m_\ell),$$

where $g^*(m)$ is the actual cost of a minimum cost path from the start node to node m. By Result 1,

$$f(m_\ell) = g^*(m_\ell) + h(m_\ell) \leq g^*(m_\ell) + h^*(m_\ell).$$

Since $g^*(m_\ell) + h^*(m_\ell) \leq UB$, then

$$f(m_\ell) \leq UB.$$

If $UB = g^*(m_\ell) + h^*(m_\ell)$, then $\boldsymbol{P}$ is an optimal path. If $f(m_\ell) < UB$, then node m_ℓ will not be deleted from list OPEN. □

APPENDIX C

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

Case 1. $\ell < k-1$. Let $\boldsymbol{v} = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, v_{\ell+1}, v_{\ell+2}, \ldots v_{n-1}) \in T(m_2)$ such that $h(m_2) = \sum_{i=\ell+1}^{n-1} (\phi^*_i - (-1)^{v_i})^2$. Since $\boldsymbol{v} \in T(m_2)$, then $\boldsymbol{v} \in T(m_1)$. Thus $\sum_{i=\ell+1}^{n-1} (\phi^*_i - (-1)^{v_i})^2 + c(m_1, m_2) \geq h(m_1)$, i.e., $h(m_2) + c(m_1, m_2) \geq h(m_1)$.

Case 2. $\ell = k-1$. $h(m_1) \leq h^*(m_1)$ and $h(m_2) = h^*(m_2)$. Since $h^*(m_1) - c(m_1, m_2) \leq h^*(m_2)$, then $h(m_1) \leq h^*(m_2) + c(m_1, m_2) = h(m_2) + c(m_1, m_2)$.

Case 3. $\ell > k-1$. $h(m_1) = h^*(m_1)$ and $h(m_2) = h^*(m_2)$. Since $h^*(m_1) - c(m_1, m_2) = h^*(m_2)$, then $h(m_1) = h(m_2) + c(m_1, m_2)$. □

APPENDIX D

Given $\boldsymbol{\phi}^*$ and a seed $\boldsymbol{c}^*$, we present an algorithm to calculate $h(m)$ for node m at level $\ell, -1 \leq \ell < k-1$, whose time complexity is $O(n)$.

First, we present this algorithm for the special case $\boldsymbol{c}^* = \boldsymbol{0}$. Then we show how this algorithm can be applied to calculate $h(m)$ for the case $\boldsymbol{c}^* \neq \boldsymbol{0}$ by modifying $\boldsymbol{\phi}^*$.

D.1 Algorithm for the Case $\boldsymbol{c}* = \boldsymbol{0}$

We will show that to calculate $h(m)$ we need to construct at most two vectors belonging to $T(m)$.

Consider a node m at level ℓ and let $\boldsymbol{P'}_m$ be the lowest cost path from the start node to node m found so far by the algorithm. Furthermore, let $\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell$ be the labels of $\boldsymbol{P'}_m$. Since $f(m) = g(m) + h(m) = \sum_{i=0}^{\ell} \left(\phi^*_i - (-1)^{\overline{v}_i}\right)^2 + h(m)$, then $h(m)$ depends only on the values of $\phi^*_{\ell+1}, \phi^*_{\ell+2}, \ldots$, and ϕ^*_{n-1}.

Let $\boldsymbol{u}_\ell = \left(u_{\ell(\ell+1)}, u_{\ell(\ell+2)}, \ldots, u_{\ell(n-1)}\right)$ be obtained by permuting the positions of $\left(\phi^*_{\ell+1}, \phi^*_{\ell+2}, \ldots, \phi^*_{n-1}\right)$ in such a manner that $u_{\ell i} \leq u_{\ell(i+1)}$ for $(\ell+1) \leq i \leq (n-2)$. We remark here that we can easily construct $\boldsymbol{u}_\ell$ from $\boldsymbol{u}_{-1}$. Furthermore, let $W_H(x)$ be the Hamming weight of $\boldsymbol{x}$.

Recall that when $\boldsymbol{c}^* = \boldsymbol{0}$,

$$T(m) = \{\boldsymbol{v} | \boldsymbol{v} = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, v_{\ell+1}, \ldots, v_{n-1}) \text{ and } W_H(\boldsymbol{v}) \in HW\},$$

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

Because of the definition of $T(m)$ we can compute $h(m)$ using $\boldsymbol{u}_\ell$ instead of $\left(\phi^*_{\ell+1}, \phi^*_{\ell+2}, \ldots, \phi^*_{n-1}\right)$.

Let $\boldsymbol{v}_s = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, v_{s(\ell+1)}, v_{s(\ell+2)}, \ldots, v_{s(\ell+w)}, v_{s(\ell+w+1)}, \ldots, v_{s(n-1)})$ and $\boldsymbol{v} = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, v_{\ell+1}, \ldots, v_{n-1})$ belong to $T(m)$ such that $W_H(\boldsymbol{v}_s) = W_H(\boldsymbol{v})$. Furthermore, let $v_{s(\ell+i)} = 1$ for $1 \leq i \leq w$ and $v_{s(\ell+i)} = 0$ for $(w+1) \leq i \leq (n-1)$. Thus

$$\boldsymbol{v}_s = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, 1, 1, \ldots, 1, 0, \ldots, 0).$$

It is easy to see that

$$\sum_{i=\ell+1}^{n-1} (u_{\ell i} - (-1)^{v_{si}})^2 \leq \sum_{i=\ell+1}^{n-1} (u_{\ell i} - (-1)^{v_i})^2.$$

As a consequence, when calculating $h(m)$ we need only to consider vectors in $T(m)$ with patterns such as $\boldsymbol{v}_s$. Thus we need to consider only a subset of $T(m), T'(m)$, such that it contains only vectors with patterns such as $\boldsymbol{v}_s$. Note that $T'(m) \neq \emptyset$.

We now consider three different patterns of $\boldsymbol{u}_\ell$.

Case 1. All components of $\boldsymbol{u}_\ell$ are negative.

In this case

$$h(m) = \sum_{i=\ell+1}^{n-1} (\phi_i^* - (-1)^{v_i})^2,$$

where $\boldsymbol{v} = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, v_{\ell+1}, \ldots, v_{n-1})$ is the vector in $T'(m)$ with maximum Hamming weight.

Case 2. All components of $\boldsymbol{u}_\ell$ are greater than or equal to zero.

In this case,

$$h(m) = \sum_{i=\ell+1}^{n-1} (\phi_i^* - (-1)^{v_i})^2,$$

where $v = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, v_{\ell+1}, \ldots, v_{n-1})$ is the vector in $T'(m)$ with minimum Hamming weight.

Case 3. $\boldsymbol{u}_\ell$ has at least one negative and one positive component.

Let w_ℓ be the number of components in $\boldsymbol{u}_\ell$ that are negative and let $\overline{w}_\ell = W_H((\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell))$. By Cases 1 and 2 we can easily show that at most two vectors in $T'(m)$ must be inspected to calculate $h(m)$. These vectors are as follows.

1) $\boldsymbol{v'} = (\overline{v}_0, \overline{v}_1, \overline{v}_\ell, v'_{\ell+1}, \ldots, v'_{n-1})$, such that $W_H(\boldsymbol{v'})$ is the largest value among all the vectors $\boldsymbol{v}$ belonging to $T'(m)$ satisfying $W_H(\boldsymbol{v}) \le \overline{w}_\ell + w_\ell$.

2) $\boldsymbol{v''} = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, v''_{\ell+1}, \ldots, v''_{n-1})$, such that $W_H(\boldsymbol{v''})$ is the smallest value among all vectors $\boldsymbol{v}$ belonging to $T'(m)$ satisfying $W_H(\boldsymbol{v}) > \overline{w}_\ell + w_\ell$.We remark here that if $\boldsymbol{u}_\ell$ belongs to Case 3, then at least one of the vectors $\boldsymbol{v'}$ or $\boldsymbol{v''}$ will always exist.

If both $\boldsymbol{v'}$ and $\boldsymbol{v''}$ exist, then

$$h(m) = \min\left\{ \sum_{i=\ell+1}^{n-1} \left(\phi_i^* - (-1)^{v'_i}\right)^2, \sum_{i=\ell+1}^{n-1} \left(\phi_i^* - (-1)^{v''_i}\right)^2 \right\}.$$

Otherwise, $h(m)$ is calculated using the vector that exists.

D.2 Algorithm for the Case $\boldsymbol{c}^* \neq \boldsymbol{0}$

Now we show that we can use the procedure presented in D.1 to calculate $h(m)$ for any $\boldsymbol{c}^* = (c_0^*, c_1^*, \ldots, c_\ell^*, c_{\ell+1}^*, \ldots, c_{n-1}^*)$. In order to differentiate the heuristic function calculated with respect to seed $\boldsymbol{0}$ and seed $\boldsymbol{c}^*$, we denote the heuristic function, calculating with respect to $\boldsymbol{0}$ by $h_{\boldsymbol{0}}$.

Consider a fictitious node m' such that a path $\boldsymbol{P}_{m'}$ from the start node to node m' has labels $\overline{v}_0 \oplus c_0^*, \overline{v}_1 \oplus c_1^*, \ldots, \overline{v}_\ell \oplus c_\ell^*$. We will show that

$$h(m) = \min_{\boldsymbol{v} \in T(m')} \left\{ \sum_{i=\ell+1}^{n-1} \left((-1)^{c_i^*} \phi_i^* - (-1)^{vi}\right)^2 \right\},$$

where $T(m') = \{\boldsymbol{v} | \boldsymbol{v} = (\overline{v}_0 \oplus c_0^*, \overline{v}_1 \oplus c_1^*, \ldots, \overline{v}_\ell \oplus c_\ell^*, v_{\ell+1}, \ldots, v_{n-1})$ and $d_H(\boldsymbol{v}, \boldsymbol{0}) \in HW\}$. Note that this value is $h_{\boldsymbol{0}}(m')$ when we assume that the received vector is $((-1)^{c_0^*}\phi_0^*, (-1)^{c_1^*}\phi_1^*, \ldots, (-1)^{c_{n-1}^*}\phi_{n-1}^*)$:

$$h(m) = \min_{\boldsymbol{v'} \in T(m)} \left\{ \sum_{i=\ell+1}^{n-1} \left(\phi_i^* - (-1)^{v'_i}\right)^2 \right\}$$

$$= \min_{\boldsymbol{v'} \in T(m)} \left\{ \sum_{i=\ell+1}^{n-1} \left((-1)^{c_i^*}\phi_i^* - (-1)^{c_i^* \oplus v'_i}\right)^2 \right\}.$$

Let $\boldsymbol{v} = \boldsymbol{c}^* \oplus \boldsymbol{v'}$. Thus $\boldsymbol{v'} = \boldsymbol{v} \oplus \boldsymbol{c}^*$. We must show that $\boldsymbol{v'} \in T(m)$ iff $\boldsymbol{v} \in T(m')$. $\boldsymbol{v'} \in T(m)$ iff $\boldsymbol{v'} = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, v'_{\ell+1}, \ldots v'_{n-1})$, and $d_H(\boldsymbol{v'}, \boldsymbol{c}^*) \in HW$ iff $\boldsymbol{v} \oplus \boldsymbol{c}^* = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, v'_{\ell+1}, \ldots, v'_{n-1})$ and $d_H(\boldsymbol{c}^* \oplus \boldsymbol{v'}, \boldsymbol{0}) \in HW$ iff $\boldsymbol{v} = (\overline{v}_0 \oplus c_0^*, \overline{v}_1 \oplus c_1^*, \ldots, \overline{v}_\ell \oplus c_\ell^*, v'_{\ell+1} \oplus c_{\ell+1}^*, \ldots, v'_{n-1} \oplus c_{n-1}^*)$, and $d_H(\boldsymbol{v}, \boldsymbol{0}) \in HW$ iff $\boldsymbol{v} \in T(m')$. Since $\boldsymbol{v'} \in T(m)$ iff $\boldsymbol{v} \in T(m')$, then we may consider minimization over vectors in $T(m')$ instead of in $T(m)$. Thus

$$h(m) = \min_{\boldsymbol{v} \in T(m')} \left\{ \sum_{i=\ell+1}^{n-1} \left((-1)^{c_i^*}\phi_i^* - (-1)^{v_i}\right)^2 \right\}.$$

Since the time complexity to find $\boldsymbol{v'}$ and $\boldsymbol{v''}$ in Case 3 is $O(n)$, we can conclude that the time complexity to calculate $h(m)$ is $O(n)$. □

APPENDIX E

Proof of the Property

Consider node m_ℓ at level $\ell, -1 \le \ell < k-2$. Furthermore, let $h(m_\ell) = \sum_{i=\ell+1}^{n-1} \left(\phi_i^* - (-1)^{v'_i}\right)^2$ where $(\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, v'_{\ell+1}, v'_{\ell+2}, \ldots, v'_{n-1}) \in T(m_\ell)$. Now consider the path $\boldsymbol{P}_{m_\ell, m_{k-2}} = (m_\ell, m_{\ell+1}, \ldots, m_{k-2})$ from node m_ℓ to node m_{k-2} at level $k-2$ whose labels are $v'_{\ell+1}, v'_{\ell+2}, \ldots, v'_{k-2}$. We now show that if $m_{\ell+1}$ is a node in this path at level $\ell+1$, then $f(m_\ell) = f(m_{\ell+1})$.

By definition $f(m_\ell) = g(m_\ell) + h(m_\ell) = g(m_\ell) + \left(\phi_{\ell+1}^* - (-1)^{v'_{\ell+1}}\right)^2 + \sum_{i=\ell+2}^{n-1} \left(\phi_i^* - (-1)^{v'_i}\right)^2 = g(m_{\ell+1}) + \sum_{i=\ell+2}^{n-1} \left(\phi_i^* - (-1)^{v'_i}\right)$. Since $(\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, v'_{\ell+1}, v'_{\ell+2}, v'_{\ell+3}, \ldots, v'_{n-1}) \in T(m_{\ell+1})$, then

$$\sum_{i=\ell+2}^{n-1} \left(\phi_i^* - (-1)^{v'_i}\right)^2 = \min_{\boldsymbol{v} \in T(m_{\ell+1})} \left\{ \sum_{i=\ell+2}^{n-1} (\phi_i^* - (-1)^{v_i})^2 \right\},$$

otherwise,

$$h(m_\ell) > \min_{\boldsymbol{v} \in T(m_\ell)} \left\{ \sum_{i=\ell+1}^{n-1} (\phi_i^* - (-1)^{v_i})^2 \right\}.$$

□

ACKNOWLEDGMENT

The authors would like to thank E. Weinman for her invaluable help in the preparation of this manuscript. Thanks are also due Prof. L. D. Rudolph for his helpful suggestions, and Prof. O. Biham for organizing the Research Experience for Undergraduates Program. In addition, the authors would like to thank the referees for their invaluable suggestions, which we believe have helped us to improve the presentation of our results. The authors gratefully acknowledge the use of the computational facilities of the Northeast Parallel Architectures Center (NPAC) at Syracuse University.

REFERENCES

[1] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform Theory*, pp. 284–287, Mar. 1974.

[2] L. D. Baumert and L. R. Welch, "Minimum-weight codewords in the (128,64) BCH Code," Jet Prop. Lab., California Inst. Technol., Pasadena, CA, DSN Progr. Rep. 42–42, Sept. and Oct. 1977.

[3] L. D. Baumert and R. J. McEliece, "Soft decision decoding of block codes," Jet Prop. Lab., California Inst. Technol., Pasadena, CA, DSN Progr. Rep. 42–47, July and Aug. 1978.

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

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

[6] R.W. D. Booth, M. A. Herro, and G. Solomon, "Convolutional coding techniques for certain quadratic residue codes," in *Proc. 1975 Int. Telemetering Conf.*, 1975, pp. 168–177.

[7] G. Brassard and P. Bratley, *Algorithmics Theory and Practice*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

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

[9] G. P. Cohen, P. J. Godlewski, and T. Y. Hwang, "Generating codewords in real space: Applications to decoding," in *Proc.* 3rd *Int. Colloquium Coding Theory and Applications*, Nov. 1988, pp. 114–122.

[10] J. H. Conway and N. J. A. Sloane, "Soft decoding techniques for codes and lattices, including the golay code and the Leech lattice," *IEEE Trans. Inform. Theory*, pp. 41–50, vol. IT-32, Jan. 1986.

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

[12] B. G. Dorsch, "A decoding algorithm for binary block codes and J-ary output channels," *IEEE Trans. on Inform. Theory*, pp. 391–394, vol. IT-20, May 1974.

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

[14] ———, "Coset codes—Part II: Binary lattices and related codes," *IEEE Trans. Inform. Theory*, pp. 1152–1187, vol. 34, Sept. 1988.

[15] Y. S. Han, C. R. P. Hartmann, and C-C. Chen, "Efficient maximum-likelihood soft-decision decoding of linear block codes using algorithm A^*," School of Comput. Inform. Sci., Syracuse Univ., Syracuse, NY 13244, Tech. Rep. SU-CIS-91-42, Dec. 1991.

[16] Y. S. Han and C. R. P. Hartmann, "Designing efficient soft-decision decoding algorithms for linear block codes using algorithm A^*," School of Comput. Inform. Sci., Syracuse Univ., Syracuse, NY 13244, Tech. Rep. SU-CIS-91-10, June 1992.

[17] T.-Y. Hwang, "Decoding linear block codes for minimizing word error rate," *IEEE Trans. Inform. Theory*, pp. 733–737, Nov. 1979.

[18] T. Kancko, T. Nishijima, H. Inazumi, and S. Hirasawa, "An efficient maximum-likelihood-decoding algorithm for linear block codes with algebraic decoder," *IEEE Trans. Inform. Theory*, to be published.

[19] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. New York: Elsevier, 1977.

[20] C. L. Mallows and N. J. A. Sloane, "An upper bound for self-dual codes," *Inform. Contr*, vol. 22, pp. 188–200, 1973.

[21] K. R. Matis and J. W. Modestino, "Reduced-search soft-decision trellis decoding of linear block codes," *IEEE Trans. Inform. Theory*, vol. IT-28, pp. 349–355, Mar. 1982.

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

[23] I. S. Reed, T. K. Truong, X. Chen, and X. Yin, "The algebraic decoding of the (41,21,9) quadratic residue code," *IEEE Trans. Inform. Theory*, vol. 38, pp. 974–986, May 1992.

[24] G. Solomon and H. C. A. van Tilborg, "A connection between block and convolutional codes," *SIAM J. Appl. Math.*, pp. 358–369, 1979.

[25] D. J. Taipale and M. B. Pursley, "An improvement to generalized-minimum-distance decoding," *IEEE Trans. Inform. Theory*, vol. 37, pp. 167–172, Jan. 1991.

[26] A. M. Tenenbaum and M. J. Augenstein, *Data Structures Using Pascal*. Englewood Cliffs, NJ: Prentice-Hall, 1986, 2nd ed.

[27] A. J. Viterbi, "Error bound for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Information Theory*, vol. IT-13, pp. 260–269, Apr. 1967.

[28] J. K. Wolf, "Efficient maximum likelihood decoding of linear block codes using a trellis," *IEEE Trans. Inform. Theory*, vol. IT-24, pp. 76–80, Jan. 1978.