A New Treatment of Priority-First Search Maximum-Likelihood Soft-Decision Decoding of Linear Block Codes

Yunghsiang S. Han, *Member, IEEE*

***Abstract*—In this correspondence we present a new method to convert the maximum-likelihood soft-decision decoding problem for linear block codes into a graph search problem where generalized Dijkstra's algorithm can still be applied to the decoding procedure. The cost assigned to every branch in the graph is based on a generalization of Wagner rule which is an equivalent form of the maximum-likelihood decoding rule used in [1]. The new decoding algorithm uses the properties of error patterns to reduce the search space.**

***Index Terms*— Decoding, Dijkstra's algorithm, linear block codes, maximum-likelihood, priority-first search, soft-decision.**

I. Introduction

The authors in [1] proposed a novel and efficient maximum-likelihood soft-decision decoding algorithm for linear block codes. The approach used there converts the decoding problem into a search problem through a graph that is a code tree for an equivalent code of the transmitted code. In a code tree, a path from the start node to a goal node corresponds to a codeword. Furthermore, every branch in the graph is assigned a cost based on a maximum-likelihood decoding rule. A generalized Dijkstra's algorithm (GDA) [2], [1] that uses a priority-first search strategy is employed to search through this graph. The purpose of the search is to find a desired path (codeword) which satisfies the maximum-likelihood decoding rule. This search is guided by an evaluation function f defined for every node in the graph to take advantage of the information provided by the received vector and codewords of the transmitted code. The algorithm maintains a list OPEN of nodes of the graph that are candidates to be expanded. The node on list OPEN with minimum values of function f is selected to expand. If the algorithm selects a goal node for expansion, it has found a desired path from the start node to the goal node. Such a path is denoted as an optimal path. Furthermore, this algorithm also keeps an upper bound on the value of f for every node in an optimal path. If the value of f for a node is larger than this upper bound, the node can be discarded. Consequently, no further search through this node is necessary. The function f defined above contains two parts: a function g and a function h. The function g for a node is obtained by summing all the branch costs encountered while constructing the path from the start node to this node; the function h for a node is an estimate of the minimum costs among all the paths from this node to goal nodes. The worst case for this decoding algorithm is to search all the nodes in the graph whose total number is $2^{k+1}-1$. Although the probability of occurrence of the worst case is rare, it is worthy to find a way to reduce the total number of nodes searched for the worst case.

In Section II we review MLD of linear block codes, and briefly describe the decoding algorithm proposed in [1]. In Section III we present a new method to convert the decoding problem into a graph search problem where GDA can still apply on the decoding procedure. The new decoding algorithm searches through error patterns instead of codewords, and in the worst case the total number of $\Sigma_{i=0}^{n-k}\binom{k+1}{i+1}$ of nodes is searched. Concluding remarks are addressed in Section IV.

Manuscript received June 8, 1986; revised February 12, 1998. This work was supported by National Science Council ROC under Grant NSC 85-2213-E-211-002. The material in this correspondence was presented at the IEEE International Symposium on Information Theory, Ulm, Germany, June 1997.

The author is with the Department of Computer Science and Information Engineering, National Chi Nan University, Taiwan, R.O.C.

Publisher Item Identifier S 0018-9448(98)06752-2.

II. Preliminaries

Let $\boldsymbol{C}$ be a binary (n,k) linear code with generator matrix $\boldsymbol{G}$, and let $\boldsymbol{c}=(c_0,c_1,\cdots,c_{n-1})$ be a codeword of $\boldsymbol{C}$ transmitted over a time-discrete memoryless channel with output alphabet $\boldsymbol{B}$. Furthermore, let $\boldsymbol{r}=(r_0,r_1,\cdots,r_{n-1})$, $r_j\in\boldsymbol{B}$ denote the received vector, and assume that $\Pr(r_j|c_i)>0$ for all $r_j\in\boldsymbol{B}$ and $c_i\in\mathrm{GF}(2)$. 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 [3]–[5]

$$\text{set } \hat{\boldsymbol{c}}=\boldsymbol{c}_\ell, \qquad \text{where } \boldsymbol{c}_\ell\in\boldsymbol{C} \text{ and}$$
$$\sum_{j=0}^{n-1}(\phi_j-(-1)^{c_{\ell j}})^2\le\sum_{j=0}^{n-1}(\phi_j-(-1)^{c_{ij}})^2, \qquad \text{for all } \boldsymbol{c}_i\in\boldsymbol{C}$$

where

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

We, therefore, may consider that the "received vector" is $\boldsymbol{\phi}=(\phi_0,\phi_1,\cdots,\phi_{n-1})$.

A code tree is a way to represent every codeword of an (n,k) code $\boldsymbol{C}$ as a path through a tree containing $n+1$ levels. In the code tree every path is totally distinct from every other path. The leftmost node is called the *start node*, which is at level -1. We denote the start node m_{start}. There are two branches, labeled by 0 and 1, respectively, that leave each node at the first k levels. After the k levels, there is only one branch leaving each node. The 2^k rightmost nodes are called *goal nodes*, which are at level $n-1$. The labels of any path from m_{start} to a goal node represent a codeword. Let $\boldsymbol{G}$ be a generating matrix of $\boldsymbol{C}$ whose first k columns form the $k\times k$ identity matrix. Furthermore, let $c_0,c_1,\cdots,c_{k-1}$ be the sequence of labels encountered when traversing a path from m_{start} to a node m at level $k-1$. Then $c_k,c_{k+1},\cdots,c_{n-1}$, the sequence of labels encountered when traversing the only path from node m to a goal node, can be obtained as follows:

$$(c_0,c_1,\cdots,c_k,c_{k+1},\cdots,c_{n-1})=(c_0,c_1,\cdots,c_{k-1})\boldsymbol{G}.$$

We now give a short description of the decoding algorithm presented in [1]. This algorithm uses the priority-first search strategy, thus avoiding traversing the entire code tree. Guided by an evaluation function f, it searches through a code tree for a code $\boldsymbol{C}^*$, which is equivalent to code $\boldsymbol{C}$. $\boldsymbol{C}^*$ is obtained from $\boldsymbol{C}$ by permuting the positions of codewords of $\boldsymbol{C}$ in such a way that the first k positions of codewords in $\boldsymbol{C}^*$ correspond to the "most reliable linearly independent" positions in the received vector $\boldsymbol{\phi}$. Let $\boldsymbol{G}^*$ be a generator matrix of $\boldsymbol{C}^*$ whose first k columns form a $k\times k$ identity matrix. In the decoding algorithm the vector $\boldsymbol{\phi}^*=(\phi_0^*,\phi_1^*,\cdots,\phi_{n-1}^*)$ 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 $\boldsymbol{G}^*$.

Now the branch costs in the code tree of $\boldsymbol{C}^*$ are specified. The cost of the branch from a node at level $t-1$ to a node at level t

is assigned the value $(\phi_t^* - (-1)^{c_t^*})^2$, where c_t^* is the label of the branch. Thus the solution of the decoding problem is converted into finding a lowest cost path from m_{start} to a goal node. Such a path will be called an optimal path.

The evaluation function f for every node m in the code tree is defined as $f(m) = g(m) + h(m)$, where $g(m)$ is the cost of the path from m_{start} to node m and $h(m)$ is an estimate of the minimum cost among all the paths from node m to goal nodes. The cost of a path is obtained by summing all the branch costs encountered while constructing this path. GDA requires that for all nodes m_i and m_j such that node m_j is an immediate successor of node m_i

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

where $c(m_i, m_j)$ is the branch cost between node m_i and node m_j. This requirement guarantees that GDA will always find an optimal path. In GDA, the next node to be expanded is one with the smallest value of f on the list of all leaf nodes (list OPEN) of the subtree constructed so far by the algorithm. Thus list OPEN must be kept ordered according to the values f of its nodes. When the algorithm chooses to expand a goal node, it is time to stop, because the algorithm has constructed an optimal path.

We now define the estimate function h given in [1]. 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 $w_0 < w_1 < \cdots < w_I$. Let $\boldsymbol{c}^*$ be a given codeword of $\boldsymbol{C}^*$. Function h is defined with respect to $\boldsymbol{c}^*$, which is called the seed of the decoding algorithm.

1) For nodes at level ℓ, with $-1 \leq \ell < k - 1$:

Let m be a node at level ℓ and let $\overline{v}_0, \overline{v}_1, \cdots, \overline{v}_\ell$ be the labels of the path $\boldsymbol{P}'_m$ from m_{start} to node m. 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,

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

Function h is defined 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 $\ell, k - 1 \leq \ell < n$:

Let m be a node at level ℓ. Function h is defined as

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

where $v_{\ell+1}^*, v_{\ell+2}^*, \cdots, v_{n-1}^*$ are the labels of the only path $\boldsymbol{P}_m$ from node m to a goal node. The estimate $h(m)$ computed here is always exact.

An algorithm to calculate $h(m)$ for node m at level $\ell, -1 \leq \ell < k - 1$, whose time complexity is $O(n)$, is presented in [1]. There, the decoding algorithm is shown to be a depth-first search-type algorithm. Thus upper bounds (UB's) on the cost of an optimal path are obtained whenever a codeword is generated. These UB's can be used to reduce the size of list OPEN. More details about this decoding algorithm can be found in [1] where authors also described other speedup techniques, such as stopping criterion and changing the seed during decoding procedure. Furthermore, the 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. We remark here that since it is rare to have more than one optimal paths in the ML decoding problem, in the remaining part of the correspondence, we assume, for simplification, that there exists a unique optimal path in the decoding problem. Furthermore, we present a result given in [1] which will be used later.

Theorem 1: Let two functions

$$f_1(m) = g(m) + h_1(m)$$

and

$$f_2(m) = g(m) + h_2(m)$$

satisfy

$$h_1(m) \leq h_2(m)$$

for every nongoal node m. Furthermore, there exists a unique optimal path. Then the GDA, using evaluation function f_2, will never expend more nodes than the algorithm using evaluation function f_1.

III. A New Evaluation Function f_{new}

In the preceding section, the decoding problem has been converted into a search problem through a graph based on the MLD rule described there. In this section, we will describe how the decoding problem can be converted into a search problem through a graph based on another equivalent form of MLD rule, which is a generalization of the Wagner rule. Furthermore, GDA can be applied to the new decoding procedure. Some speedup techniques which can further reduce the search space during the decoding procedure are also presented. The techniques use the properties of the error patterns and some of them are analogous to those presented in [1]. The worst case for new decoding algorithm will be shown to be bounded by $\Sigma_{i=0}^{n-k} \binom{k+1}{i+1}$ instead of $2^{k+1} - 1$ in [1]. Hence the decoding algorithm is suitable for both high-rate codes and low-rate codes.

Let $\boldsymbol{y} = (y_0, y_1, \cdots, y_{n-1})$ be the hard decision of $\boldsymbol{\phi}^*$. That is,

$$y_i = \begin{cases} 1, & \text{if } \phi_i^* < 0 \\ 0, & \text{otherwise.} \end{cases}$$

Furthermore, let $\boldsymbol{s} = \boldsymbol{y}\boldsymbol{H}^{*T}$ be the syndrome of $\boldsymbol{y}$, where $\boldsymbol{H}^*$ is a parity-check matrix for code $\boldsymbol{C}^*$. Let $E(\boldsymbol{s})$ be the collection of all error patterns whose syndrome is $\boldsymbol{s}$. The MLD rule that is a generalization of the Wagner rule is as follows [6]:

$$\text{set } \hat{\boldsymbol{c}} = \boldsymbol{y} \oplus \boldsymbol{e}_\ell, \quad \text{where } \boldsymbol{e}_\ell \in E(\boldsymbol{s}) \text{ and}$$
$$\sum_{j=0}^{n-1} e_{\ell j} |\phi_j^*| \leq \sum_{j=0}^{n-1} e_j |\phi_j^*|, \qquad \text{for all } \boldsymbol{e} \in E(\boldsymbol{s}). \tag{2}$$

We now specify the branch cost from a node at level $t - 1$ to a node at level t in the code tree of $\boldsymbol{C}^*$ as the value $|\phi_t^*|(y_t \oplus c_t^*)$, where c_t^* is the label of the branch. Thus according to inequality (2), the solution of the decoding problem is converted into finding a lowest cost path from m_{start} to a goal node. In this case, the decoding algorithm searches through the error patterns in $E(\boldsymbol{s})$. Now we define an evaluation function f_{new} for every node m in the code tree as $f_{\text{new}}(m) = g_{\text{new}}(m) + h_{\text{new}}(m)$. Let $\overline{v}_0, \overline{v}_1, \cdots, \overline{v}_\ell$ be the labels of the path $\boldsymbol{P}'_m$ from m_{start} to node m. Then

$$f_{\text{new}}(m) = \sum_{i=0}^{\ell} |\phi_i^*|(y_i \oplus \overline{v}_i) + h_{\text{new}}(m).$$

We now define our estimate function $h_{\rm new}$, which satisfies inequality (1). Since the decoding algorithm searches through the error patterns in $E(\boldsymbol{s})$ and $\boldsymbol{C}^*$ is obtained by ordering the positions of the codewords in $\boldsymbol{C}$ according to the reliability of positions of the received vector, we must use properties of the error patterns in $E(\boldsymbol{s})$ that are invariant under any permutation of the positions of the codewords to define function $h_{\rm new}$. Let $HW = \{w_i | 0 \le i \le I\}$ be the set of all distinct Hamming weights that codewords of $\boldsymbol{C}$ may have. Furthermore, assume $w_0 < w_1 < \cdots < w_I$. Thus the Hamming distance between any two codewords of $\boldsymbol{C}^*$ must belong to HW. Consequently, the Hamming distance between any two error patterns in $E(\boldsymbol{s})$ also belongs to HW.

Let $\boldsymbol{e}$ be a given error pattern of $E(\boldsymbol{s})$. Our function $h_{\rm new}$ is defined with respect to $\boldsymbol{e}$, which is called the seed of the decoding algorithm.

1) For nodes at level ℓ, with $-1 \le \ell < k-1$:

Let m be a node at level ℓ, and let $\overline{v}_0, \overline{v}_1, \cdots, \overline{v}_\ell$ be the labels of the path $\boldsymbol{P}'_m$ from $m_{\rm start}$ to node m. We now construct the set $V(m)$ of all binary n-tuples $\boldsymbol{v}$ such that their first $\ell+1$ entries are the labels $\overline{v}_i \oplus y_i, 0 \le i \le \ell$, and $d_H(\boldsymbol{v}, \boldsymbol{e}) \in HW$, where $d_H(\boldsymbol{x}, \boldsymbol{y})$ is the Hamming distance between $\boldsymbol{x}$ and $\boldsymbol{y}$. That is,

$$V(m) = \{\boldsymbol{v} | \boldsymbol{v} = (\overline{v}_0 \oplus y_0, \overline{v}_1 \oplus y_1, \cdots, \overline{v}_\ell \oplus y_\ell, v_{\ell+1}, \cdots, v_{n-1}) \text{ and } d_H(\boldsymbol{v}, \boldsymbol{e}) \in HW\}.$$

Note that $V(m) \ne \emptyset$. This can be seen by considering the binary k-tuple $\boldsymbol{u} = (\overline{v}_0, \overline{v}_1, \cdots, \overline{v}_\ell, 0, \cdots, 0)$ and

$$\overline{\boldsymbol{e}} = (\overline{v}_0 \oplus y_0, \overline{v}_1 \oplus y_1, \cdots, \overline{v}_{n-1} \oplus y_{n-1}) \in V(m)$$

where

$$\boldsymbol{u} \cdot \boldsymbol{G}^* = (\overline{v}_0, \overline{v}_1, \cdots, \overline{v}_\ell, \overline{v}_{\ell+1}, \cdots, \overline{v}_{n-1}).$$

We now define function $h_{\rm new}$ as

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

2) For nodes at level $\ell, k-1 \le \ell < n$:

Let m be a node at level ℓ. We define function $h_{\rm new}$ as

$$h_{\rm new}(m) = \sum_{i=\ell+1}^{n-1} |\phi_i^*| (y_i \oplus v_i^*)$$

where $v_{\ell+1}^*, v_{\ell+2}^*, \cdots, v_{n-1}^*$ are the labels of the only path $\boldsymbol{P_m}$ from node m to a goal node. The estimate $h_{\rm new}(m)$ computed here is always exact. Note that if node m is a goal node, then $h_{\rm new}(m) = 0$.

It is very important that the calculation of $h_{\rm new}$ cannot be heavily time-consuming. In order for GDA using function $f_{\rm new}$ to be feasible, there must be a linear time complexity (with respect to n) algorithm to calculate $h_{\rm new}$. Next we show that the algorithm presented in [1] to calculate $h(m)$ with time complexity of $O(n)$ can be used to calculate $h_{\rm new}(m)$.

Theorem 2: Let $\boldsymbol{c}^*$ be a given seed for GDA using function f and $\boldsymbol{c}^* \oplus \boldsymbol{y}$ be the given seed for GDA using function $f_{\rm new}$. Then

$$h(m) = \sum_{i=\ell+1}^{n-1} (\phi_i^* - (-1)^{v_i})^2 \text{ iff}$$
$$h_{\rm new}(m) = \sum_{i=\ell+1}^{n-1} |\phi_i^*| (y_i \oplus v_i).$$

The proof of Theorem 2 is given in Appendix A.

By Theorem 2, the algorithm to calculate $h(m)$ can be used to find $h_{\rm new}(m)$, and *vice versa*. Thus the algorithm to calculate $h(m)$ presented in [1] with time complexity of $O(n)$ can be used to calculate $h_{\rm new}(m)$. This results in the following theorem.

Theorem 3: There exists a time complexity of $O(n)$ algorithm to calculate $h_{\rm new}(m)$.

It can be shown that not only can an algorithm be used to calculate both estimate functions, but also GDA using these two functions results in the same performance. That is, even though the function $f_{\rm new}$ is defined according inequality (2), GDA using function $f_{\rm new}$ will expand the same set of nodes in the code tree as that using function f defined in Section II. Therefore, the proposed new decoding algorithm is at least as good as that given in [1].

Theorem 4: Let $f(m) = g(m) + h(m)$ and $f_{\rm new} = g_{\rm new}(m) + h_{\rm new}(m)$. Then GDA using evaluation function $f_{\rm new}$ will expand the same set of nodes as that using evaluation function f.

The proof of Theorem 4 is given in Appendix B.

We have shown that the GDA using the new evaluation function $f_{\rm new}$ has the same performance as that using evaluation function f. As mentioned in [1], the estimate function $h_{\rm new}(m)(h(m))$ can help GDA to reduce the search space even though GDA spends some computation power on calculating it. In order to simplify the decoding algorithm one may not want to use $h_{\rm new}(h)$ to reduce the search space and only use $g_{\rm new}(g)$ to search through the code tree. Under this circumstance, we show that GDA using $g_{\rm new}$ will expand no more nodes than that using g.

Lemma 1: Let $S = \{0, 1, 2, \cdots, n\}$. It is clear that S is a super set of HW. For any given seed of GDA

$$h(m) = \sum_{i=\ell+1}^{n-1} (|\phi^*| - 1)^2 = h_s(m)$$

and

$$h_{\rm new}(m) = 0$$

if $h(m)$ and $h_{\rm new}(m)$ are defined with respect to S.

We omit the proof of Lemma 1 for simplification.

Let $f_s(m) = g(m) + h_s(m)$ and $f_{new_s}(m) = g_{\rm new}(m)$. By Lemma 1 and Theorem 4 we can conclude the following result.

Theorem 5: GDA using evaluation function $f_{{\rm new}_s}$ expands the same set of nodes as that using evaluation function f_s.

Since $g(m) \le f_s(m)$ for every nongoal node m, by Theorem 1, GDA using evaluation function $f_{{\rm new}_s}$ will expand no more nodes than that using evaluation function g.

Simulation results for GDA using $f_{{\rm new}_s}$ and using $f = g$ are given in Table I. These results were obtained by simulating 10 000 samples for each SNR (signal-to-noise ratio) for the $(104, 52)$ binary extended quadratic residue code when the code is transmitted over the additive white Gaussian noise channel. For those samples tried, the new decoding algorithm is much better than that presented in [1] when no estimate functions are used. Furthermore, for GDA using $f = g$, when SNR was under 6 dB, the computer crashed due to lack of memory. Therefore, if GDA is using $f = g$, then it will become impractical with SNR under 6 dB for the code $(104, 52)$. We remark here that an equivalent form of GDA using $f_{{\rm new}_s}$ was independently proposed very recently in [7] where computer simulation results were also given for several codes.

TABLE I
THE AVERAGE AND MAXIMUM NUMBER OF NODES OPENED DURING THE DECODING OF (104, 52) CODE

GDA using $f = g$

γ_b	6 dB		7 dB		8 dB		9 dB	
	max	ave	max	ave	max	ave	max	ave
$N(\phi)$	11,396,950	92,680	1,153,378	17,392	251,226	4,666	93,105	1,669

GDA using f_{new},

γ_b	5 dB		6 dB		7 dB		8 dB		9 dB	
	max	ave	max	ave	max	ave	max	ave	max	ave
$N(\phi)$	218,226	915	11,477	194	1,376	115	95	62	57	54

$N(\phi)$= the number of nodes opened during the decoding of ϕ

TABLE II
WORST CASE DECODING COMPLEXITY

Code(n,k)	N^a_{old}	N^b_{new}
Golay(24,12)	8,191	8,191
BCH(15,11)	4,095	1,585
BCH(31,26)	$\approx 1.34 \times 10^8$	397,593
BCH(63,57)	$\approx 2.88 \times 10^{17}$	$\approx 3.5 \times 10^8$

[a]GDA given in [1].
[b]GDA using f_{new}.

The decoding algorithm searches the code tree only up to level $k-1$ since $\boldsymbol{G}^*$ can be used to construct the only path from any node m at level $k-1$ to a goal node. Thus the cost of the path can be calculated and used as an upper bound on the cost of the optimal path. Any node m on list OPEN with or equal to this bound can be eliminated from list OPEN.

A criterion can be used to check if the search may be stopped or not. The criterion is as follows.

Criterion 1: If

$$h_{\text{new}}(m_{\text{start}}) = \sum_{i=0}^{n-1} |\phi^*| e_i$$

where the seed of the decoding algorithm is $\boldsymbol{e}$, then $\boldsymbol{e}$ is the error pattern which satisfies inequality (2).

This criterion follows directly from the definition of h_{new}. Thus during the decoding procedure, if any seed $\boldsymbol{e}$ satisfies Criterion 1, then $\boldsymbol{e} \oplus \boldsymbol{y}$ is the desired codeword.

The seed $\boldsymbol{e}$ does not need to be fixed during the decoding procedure. It may be changed according to any rule which will help to reduce the search space. Under these circumstances, the decoding algorithm has an adaptive decoding procedure. The values of function h_{new} of the nodes on list OPEN will not be recalculated with respect to the new seed in order to save computation power. Only new open nodes are calculated with respect to the new seed. As proved in [1], the adaptive decoding procedure still finds the optimal path.

Since GDA using f_{new} searches through all possible error patterns, we may use the properties of error patterns to reduce the search space. If an error pattern has Hamming weight which is greater than $n-k$, then this error pattern can be eliminated from the search space [6], [8]. That is, when GDA using f_{new} searches for the optimal path, the path which corresponds to an error pattern with Hamming weight greater than $n-k$ can be eliminated from list OPEN. Consequently, the worst case of the search space for GDA will be bounded by

$$\sum_{j=0}^{k} \sum_{i=0}^{n-k} \binom{j}{i}$$

which equals to

$$\sum_{i=0}^{n-k} \binom{k+1}{i+1}.$$

When $n-k > k$, the terms $\binom{k+1}{k+2}, \cdots,$ and $\binom{k+1}{n-k+1}$ are all zeros. Therefore, in this case,

$$\sum_{i=0}^{n-k} \binom{k+1}{i+1} = \sum_{i=0}^{k} \binom{k+1}{i+1} = 2^{k+1} - 1.$$

The worst case decoding complexity of the GDA using f_{new} and the GDA given in [1] for some codes are given in Table II. By the results given, the new decoding algorithm is quite suitable for both high-rate and low-rate codes.

Several trellis-based decoding algorithms were proposed [9], [8]. The method proposed in [9] is to apply the Viterbi algorithm on a trellis of the code and that in [8] on a minimal trellis representation of the code. Table III presents for each code studied the number of nodes opened for these trellis-based decoding algorithms. All of these are much greater than the average number of nodes opened by the proposed decoding algorithm shown in the table. In [8], a punctured minimal trellis technique was proposed to further reduce the number of nodes opened in a minimal trellis presentation of the code; however, this technique is suitable only for high-rate codes.

Another maximum-likelihood soft-decision decoding algorithm using an algebraic decoder was recently reported [10]. This approach is a generalization of Chase algorithm and it performs MLD rule.

TABLE III
COMPARISON OF THE AVERAGE DECODING COMPLEXITY FOR TRELLIS-BASED DECODING ALGORITHMS

Code (n, k)	Wolf's algorithm [9]	algorithm given in [8]	the proposed algorithm		
			$1\ dB$	$4\ dB$	$7\ dB$
Golay $(24, 12)$	$\approx 16,380^a$	$\approx 3,580^a$	16	1	1
QR $(48, 24)$	$\approx 67,108,860^a$	$\approx 860,156^a$	826	9	1

^{a}from [7].

TABLE IV
THE DECODING COMPLEXITY OF DECODING OF (128, 64) CODE

γ_b	$5\ dB$		$5.5\ dB$		$6\ dB$		$6.5\ dB$		$7\ dB$	
	max	ave	max	ave	max	ave	max	ave	max	ave
$N(\phi)^a$	102,364	18	3,069	3	2,638	1	416	1	204	1
$N_{AD}(\phi)^b$	2,097,152	283	32,768	4	1,024	1	5	1	1	1

$^a N(\phi)=$ the number of nodes opened during the decoding of ϕ by the proposed decoding algorithm.
$^b N_{AD}(\phi)=$ the number of algebraic decoder used during the decoding of ϕ (given in [10]) by the decoding algorithm given in [10].

Table IV shows the number of algebraic decoder used by this method for the $(128, 64)$ extended BCH code. One may see that the decoding complexity of the proposed decoding algorithm is much smaller than that of the algorithm given in [10] at 5.0 and 5.5 dB. Furthermore, the decoding algorithm given in [10] can only work on codes for which algebraic decoders exist; however, the proposed decoding algorithm can work on all linear block codes.

IV. CONCLUSIONS

In this correspondence we present a new method to convert the decoding problem into a graph-search problem based on the generalization of Wagner rule. GDA can be applied on the graph to reduce the search space. We show that the new decoding algorithm opens less nodes than that presented in [1] for the worst cases. From simulation results, the new decoding algorithm has about 1000 computation gain over the old one when no estimate functions are used. Although we concentrate only on the case that the graph is a code tree, the graph can also be a trellis. The decoding algorithm must be modified to check for the repeated nodes if we apply GDA to a trellis. The branch costs assigned to the code tree which are presented in Section II may be replaced with the value $-(-1)^{c_t^*}\phi_t^*$ to save computation power. The designed heuristic function can not violate the requirement of inequality (1) in order to guarantee that GDA will find an optimal path. For example, when $f = g$, the branch cost cannot be changed to the value $-(-1)^{c_t^*}\phi_t^*$ since it violates the requirement of inequality (1).

APPENDIX A
PROOF OF THEOREM 2

Let $\boldsymbol{c}^*$ be a given seed of GDA using f and $\boldsymbol{e} = \boldsymbol{c}^* \oplus \boldsymbol{y}$ for that using f_{new}. Furthermore, let $\boldsymbol{v} = (\overline{v}_0, \overline{v}_1, \cdots, \overline{v}_\ell, v_{\ell+1}, \cdots, v_{n-1})$ be the vector such that

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

We prove that

$$h_{\text{new}}(m) = \sum_{i=\ell+1}^{n-1} |\phi_i^*|(v_i \oplus y_i).$$

Since $d_H(\boldsymbol{v}, \boldsymbol{c}^*) \in HW$, then $d_H(\boldsymbol{v} \oplus \boldsymbol{y}, \boldsymbol{c}^* \oplus \boldsymbol{y}) \in HW$. Assume that there exists another error pattern $\boldsymbol{x} \in E(\boldsymbol{s})$ and $\boldsymbol{x} \in \boldsymbol{V}(m)$ such that

$$\sum_{i=\ell+1}^{n-1} |\phi_i^*|x_i < \sum_{i=\ell+1}^{n-1} |\phi_i^*|(v_i \oplus y_i). \tag{3}$$

Since $\boldsymbol{x} \in V(m)$, $\boldsymbol{x}$ must be the form of

$$(\overline{v}_0 \oplus y_0, \overline{v}_1 \oplus y_1, \cdots, \overline{v}_\ell \oplus y_\ell, x_{\ell+1}, \cdots, x_{n-1})$$

and $(\boldsymbol{x}, \boldsymbol{e}) \in HW$. Since $(\boldsymbol{x}, \boldsymbol{e}) \in HW$, then

$$(\boldsymbol{x} \oplus \boldsymbol{y}, e \oplus \boldsymbol{y}) = (\boldsymbol{x} \oplus \boldsymbol{y}, \boldsymbol{c}^*) \in HW$$

where

$$\boldsymbol{x} \oplus \boldsymbol{y} = (\overline{v}_0, \overline{v}_1, \cdots, \overline{v}_\ell, x_{\ell+1} \oplus y_{\ell+1}, \cdots, x_{n-1} \oplus y_{n-1}).$$

Thus $\boldsymbol{x} \oplus \boldsymbol{y} \in T(m)$. If we multiply inequality (3) by 2 and evaluate all possible values among y_i, v_i, and x_i, the inequality equals to

$$\sum_{i=\ell+1}^{n-1} \phi_i^*((-1)^{y_i} - (-1)^{x_i \oplus y_i}) < \sum_{i=\ell+1}^{n-1} \phi_i^*((-1)^{y_i} - (-1)^{v_i}). \tag{4}$$

Inequality (4) can be further simplified to

$$-2\sum_{i=\ell+1}^{n-1} \phi_i^*(-1)^{x_i \oplus y_i} < -2\sum_{i=\ell+1}^{n-1} \phi_i^*(-1)^{v_i}$$

which equals

$$\sum_{i=\ell+1}^{n-1} (\phi_i^* - (-1)^{x_i \oplus y_i})^2 < \sum_{i=\ell+1}^{n-1} (\phi_i^* - (-1)^{v_i})^2. \tag{5}$$

By the definition of function h, inequality (5) contradicts the assumption that

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

The proof of the rest of theorem is similar to that above.

Appendix B
Proof of Theorem 4

Let node m be a node at level ℓ in the code tree and the labels of path $\boldsymbol{P}'_m$, the path from m_{start} to node m, are $\overline{v}_0, \overline{v}_1, \cdots, \overline{v}_\ell$. Furthermore, let $v_0, v_1, \cdots, v_{n-1}$ be the labels of an optimal path from m_{start} to a goal node. Now we consider $f(m) = g(m) + h(m)$ and $f_{\text{new}}(m) = g_{\text{new}}(m) + h_{\text{new}}(m)$. By Theorem 2, we may assume that

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

and

$$h_{\text{new}}(m) = \sum_{i=\ell+1}^{n-1} |\phi_i^*|(v_{s_i} \oplus y_i).$$

According to the search rule in GDA, node m will be expanded during the search procedure if and only if

$$f(m) \le \sum_{i=0}^{n-1} (\phi_i^* - (-1)^{v_i})^2 \tag{6}$$

if GDA is using evaluation function f;

$$f_{\text{new}}(m) \le \sum_{i=0}^{n-1} |\phi_i^*|(y_i \oplus v_i) \tag{7}$$

if GDA is using evaluation function f_{new}.

Since

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

then

$$f(m) \le \sum_{i=0}^{n-1} (\phi_i^* - (-1)^{v_i})^2$$

is equal to

$$\sum_{i=0}^{\ell} (\phi_i^* - (-1)^{\overline{v}_i})^2 + \sum_{i=\ell+1}^{n-1} (\phi_i^* - (-1)^{v_{s_i}})^2 \le \sum_{i=0}^{n-1} (\phi_i^* - (-1)^{v_i})^2. \tag{8}$$

Inequality (8) can be further simplified to

$$\sum_{i=0}^{\ell} \phi_i^*((-1)^{v_i} - (-1)^{\overline{v}_i}) - \sum_{i=\ell+1}^{n-1} \phi_i^*((-1)^{v_{s_i}} - (-1)^{v_i}) \le 0. \tag{9}$$

If we multiple inequality (9) by $1/2$ and evaluate all possible values among y_i, v_i, v_{s_i}, and $\overline{v}_i$, the inequality equals

$$\sum_{i=0}^{\ell} |\phi_i^*|[(y_i \oplus \overline{v}_i) - (y_i \oplus v_i)] - \sum_{i=\ell+1}^{n-1} |\phi_i^*|[(y_i \oplus v_i) - (y_i \oplus v_{s_i})] \le 0. \tag{10}$$

Since

$$f_{\text{new}}(m) = \sum_{i=0}^{\ell} |\phi_i^*|(y_i \oplus \overline{v}_i) + \sum_{i=\ell+1}^{n-1} |\phi_i^*|(v_{s_i} \oplus y_i)$$

then inequality (10) equals inequality (7). Since inequality (6) equals inequality (7), we can claim that GDA using evaluation function f_{new} will expand the same set of nodes as that using evaluation function f.

Acknowledgment

The author would like to thank the reviewer for his valuable suggestions, which he believes have helped him to improve the presentation of this correspondence.

References

[1] Y. S. Han, C. R. P. Hartmann, and C.-C. Chebn, "Efficient priority-first search maximum-likelihood soft-decision decoding of linear block codes," *IEEE Trans. Inform. Theory*, vol. 39, pp. 1514–1523, Sept. 1993.

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

[3] Y. Be'ery and J. Snyders, "Optimal soft decsion block decoders based on fast hadamrad transform," *IEEE Trans. Inform. Theory*, vol. 42, pp. 355–364, 1986.

[4] A. Vardy and Y. Be'ery, "More efficient soft-decision decoding of the golay codes," *IEEE Trans. Inform. Theory*, vol. 37, pp. 667–672, 1991.

[5] ——, "Bit-level soft decision decoding of Reed-Solomon codes," *IEEE Trans. Commun.*, vol. 39, pp. 440–445, 1991.

[6] J. Snyders and Y. Be'ery, "Maximum likelihood soft decoding of binary block codes and decoders for the golay codes," *IEEE Trans. Inform. Theory*, vol. 35, pp. 963–975, Sept. 1989.

[7] L. Ekroot and S. Dolinar, "A* decoding of block codes," *IEEE Trans. Commun.*, vol. 44, pp. 1052–1056, Sept. 1996.

[8] Y. Berger and Y. Be'ery, "Soft trellis-based decoder for linear block codes," *IEEE Trans. Inform. Theory*, vol. 40, pp. 764–773, May 1994.

[9] 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.

[10] 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*, vol. 40, pp. 320–327, Mar. 1994.