IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 4, 2018, accepted August 2, 2018, date of publication August 21, 2018, date of current version September 7, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2866396

An Improved A^* Decoding Algorithm With List Decoding

BIN XU, CHENHAO YING, AND YUAN LUO

Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China

Corresponding author: Yuan Luo (yuanluo@sjtu.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grant 61571293.

ABSTRACT Comparing with hard decision decoding algorithms, soft decoding has a lower probability of bit error but a higher computational complexity. As a maximum-likelihood soft decoding method, the A^* algorithm is the most basic and widely used to minimize bit error probability. However, its average computational complexity strongly depends on a seed codeword and a heuristic function utilized during the decoding process. To efficiently reduce the computational complexity while maintaining the decoding accuracy theoretically and practically, this paper proposes an improved A^* decoding algorithm consisting of two phases. The first phase applies the greedy list decoding to the linear block code to obtain a seed codeword. According to the seed, the second phase applies the improved A^* algorithm to obtain the final decoding output. The heuristic function used in the A^* algorithm is modified in two aspects: 1) use more information of partial decoded symbols to improve the accuracy of the function and 2) take advantage of Hamming distance to reduce the search space. Simulations on the $RM(5, 2)$ Reed–Muller codes and $[128, 64]$ binary extended BCH code show that this improved A^* algorithm is more efficient in average decoding complexity than many other algorithms while maintaining the decoding accuracy.

INDEX TERMS Error correcting coding, list decoding, A^* algorithm, computational complexity.

I. INTRODUCTION

Error correcting coding is a key method to guarantee reliable communication, which has been studied for a long time [1]. Compared with hard decision decoding algorithms [2], soft decision decoding algorithms [3] have a lower probability of bit error for block codes but higher computational complexity. To obtain a lower bit error probability and a lower computational complexity, the maximum likelihood soft decision decoding algorithms have been extensively studied in coding theory [4]–[6].

List decoding is an alternative to unique decoding of error-correcting codes for large coding rates with low error probability. For example, Guruswami *et al.* [7] and [8] used list decoding to solve problems about Reed-Muller codes and Reed-Solomon codes. Furthermore, some researchers have converted decoding schemes into a graph search problem on a trellis of the code [9]–[11]. The A^* algorithm is an artificial intelligence tree search algorithm used to search for a path in a graph. Nilsson [12] described the algorithm as a heuristic graph search procedure and showed that the algorithm always can obtain an optimal path. Han *et al*. [13] extended it to decode the block codes and proved that it is a maximum likelihood soft decision decoding algorithm. Meanwhile, Ekroot [14] further studied the properties of the A^* decoding algorithm. By considering sums of parity bits which can be decided by part of the message bits, A^* algorithm can reduce the number of search tree edges [15]. In [16], A^* algorithm has shown good performance for maximum likelihood decoding of tailbiting codes.

For a decoding algorithm, time and space complexity are two measurements to judge whether the algorithm is efficient [17]. The time complexity is always considered first and this paper is no exception. To determine the computational complexity of the A^* decoding. Han *et al.* [18] used an average complexity. Unlike the worst complexity, the average complexity is the average amount of computational resources (typically time) used by the algorithm over all possible inputs.

To construct the heuristic function which was indispensable for implementing the A^* algorithm to search the codeword path, Han *et al*. [13] used the property of the Hamming distance among codewords and defined the function by using a seed codeword. It is shown that if the seed codeword is closer to the correct codeword or the heuristic function is more accurate, the computational complexity will be lower.

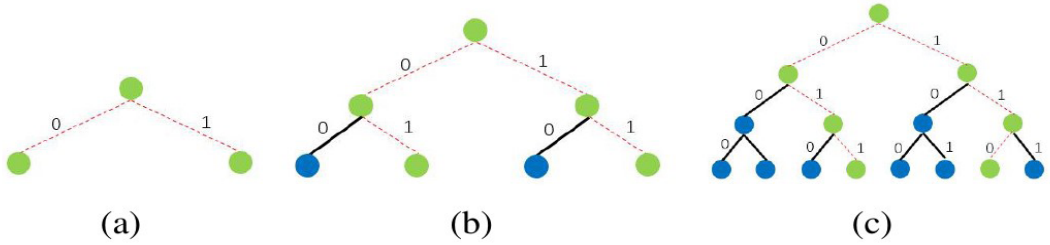


FIGURE 1. List decoding with list size $L = 2$, where reserved paths contain green nodes and red dashed edges, while deleted paths contain blue nodes and black solid edges. And part(a) to part(c) are the procedures of list decoding step by step.

Therefore, the computational complexity of A^* depends on the seed codeword and the heuristic function. For the above reasons, this paper proposes an improved A^* decoding algorithm with two phases. The first phase applies the list decoding to the linear block code to obtain a more accurate seed codeword rather than a stochastic one. The second phase employs the A^* algorithm by using a modified heuristic function. In particular, the heuristic function is modified in two aspects: 1. use more information of partial decoded symbols to improve the accuracy of the function; 2. take advantage of Hamming distance to reduce the search space. Simulations on the $RM(5, 2)$ code and $[128, 64]$ binary extended BCH code show that the proposed algorithm is much more efficient in computational complexity while maintaining the decoding accuracy.

The rest of this paper is organized as follows. Section II gives the notions of the maximum likelihood decoding, list decoding and A^* algorithm. Section III presents an improved A^* algorithm. Section IV simulates the new algorithm on the Reed-Muller code and $[128, 64]$ binary extended BCH code and provides a comparison with other decoding algorithms on the computational complexity and the bit error probability. Section V concludes this paper.

II. PRELIMINARIES AND NOTATIONS

This section gives the notion of maximum likelihood decoding rule and then provides brief reviews of the list decoding algorithm and the A^* algorithm. Let $\boldsymbol{C}$ denote a binary $[n, k]$ linear block code with generator matrix $\boldsymbol{G}$ and parity check matrix $\boldsymbol{H}$, and let $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$ denote a codeword.

A. MAXIMUM LIKELIHOOD DECODING

The popular maximum likelihood decoding rule over a time-discrete memoryless channel can be formulated as follows, where $\boldsymbol{c}$ is the transmitted codeword, and $\boldsymbol{r} = (r_0, r_1, \ldots r_{n-1})$ is the received vector. The maximum likelihood estimation of $\boldsymbol{c}$ is determined to be $\boldsymbol{c}_l = (c_{l0}, c_{l1}, \ldots, c_{l(n-1)}) \in \boldsymbol{C}$ if

$$\prod_{j=0}^{n-1} Pr(r_j|c_{lj}) \geq \prod_{j=0}^{n-1} Pr(r_j|c_{ij}) \tag{1}$$

for all $\boldsymbol{c}_i = (c_{i0}, c_{i1}, \ldots, c_{i(n-1)}) \in \boldsymbol{C}$. For computation convenience, the bit log-likelihood ratio of r_j was introduced as [19]

$$\phi_j = \ln \frac{Pr(r_j|0)}{Pr(r_j|1)}, \tag{2}$$

where $\boldsymbol{\phi} = (\phi_0, \phi_1, \ldots, \phi_{n-1})$. Then, formula (1) can be rewritten as [19]

$$\sum_{j=0}^{n-1} (\phi_j - (-1)^{c_{lj}})^2 \leq \sum_{j=0}^{n-1} (\phi_j - (-1)^{c_{ij}})^2. \tag{3}$$

This shows that the maximum likelihood decoding rule minimizes the bit error probability when the input probabilities of codewords in $\boldsymbol{C}$ are equal.

After introducing the maximum likelihood decoding rule, the list decoding algorithm and the A^* algorithm are presented in the next subsections.

B. LIST DECODING ALGORITHM

The list decoding applied in our algorithm is a breadth-first search algorithm on a code tree. As shown in Fig. 1, at each level of the tree, we only reserve the L (the list size) most promising nodes and delete others, which satisfies $L \leq 2^k$. If the number of nodes are less than L, there is no need to delete. Therefore, the list decoding in our algorithm is greedy. The L nodes at each level are called *reserved nodes* (the green nodes) and the paths from the root node to the reserved nodes are called *reserved paths* (the red dashed path). A simple example of our list decoding algorithm is illustrated in Fig. 1, where $L = 2$ and 0, 1 are the labels of edges.

Part (a) of Fig. 1: Algorithm starts by visiting and reserving nodes, which results in the two ($\leq L$) most promising paths (the red dashed paths). Part (b): Visit all four children of the reserved nodes in Part (a). Four paths are obtained by linking with parent nodes, then prune the four paths into the $L(= 2)$ most promising paths. Part (c): Continue searching at the two paths by visiting the children of the reserved nodes obtained from Part (b), then prune four paths into the best $L(= 2)$ paths again. At the end of the algorithm, choose one path from the L as the final decoding output. Actually, this chosen path may not be optimal, but it can be used to reduce the computational complexity of the second phase.

C. A^* DECODING ALGORITHM

As mentioned in Section I, the A^* decoding algorithm is an efficient maximum likelihood soft decision decoding algorithm for linear block codes [13]. For a code tree of $\boldsymbol{C}$, every node at the last layer of this tree is denoted as a goal node. For each visited node m, the **cost function** $g(m)$ is regarded as the actual cost of the path from the start node to m, and **heuristic function** $h(m)$ estimates the cost of the minimum-cost path from m to a goal node. The algorithm stores the path $\boldsymbol{P}_m$ from the start node to node m according to the **estimation function** $f(m) = g(m) + h(m)$.

In the storing procedure of a given code tree, the A^* algorithm maintains and updates two lists of nodes, namely, list CLOSED and list OPEN. List OPEN contains the set of nodes that were visited but not expanded yet. List CLOSED contains the set of nodes that were visited and also expanded. These visiting and expanding conditions are two basic notions in breadth-first search algorithm. In fact, the algorithm selects node m from list OPEN according to the minimum $f(m)$, and then expands and moves this node into list CLOSED. Finally add the immediate successors of node m into list OPEN for the preparation of next step. When the algorithm arrives at goal node, it obtains a path with the smallest cost, the labels of which are regarded as the decoding result in the end.

III. NEW DECODING ALGORITHM

After reviewing some basic knowledge, this section will propose an improved A^* algorithm. As introduced previously, the new algorithm applies the list decoding in the first phase to obtain a more accurate seed codeword, and employs the A^* algorithm in the second to obtain the final decoding result. This section has two theorems. Theorem 1 proves that the improved A^* algorithm is a maximum likelihood decoding for all linear block codes. After obtaining the computational complexity of the first phase, Theorem 2 gives the computational complexity of the second phase. At the end of this section, it is concluded that the new decoding algorithm reduces the computational complexity without reducing the decoding accuracy.

A. THE FIRST PHASE

According to the interpretation in Subsection II-B, the list decoding applied in the first phase to obtain a more accurate seed codeword is a breadth-first search algorithm. Let $\boldsymbol{c}_d$ denote this more accurate seed codeword. To apply the list decoding algorithm, two questions should be considered: (1) How to construct a code tree for the linear block code; (2) How to decide the most promising path.

1) CODE TREE

For an $[n, k]$ linear block code $\boldsymbol{C}$ with generator matrix $\boldsymbol{G}$ and parity check matrix $\boldsymbol{H}$, it is known that list decoding is not a maximum likelihood decoding, i.e., the algorithm can not ensure the minimization of the bit error probability. For reducing the error probability, our code tree is derived from a new generator matrix $\boldsymbol{G}^*$ by using the original $\boldsymbol{H}$. The steps are shown as follows.

1) **$\boldsymbol{H}_1$ and $\boldsymbol{\phi}_1$.**
   Linearly combine the rows and rearrange the columns of $\boldsymbol{H}$ to obtain a new matrix $\boldsymbol{H}_1$, which can be expressed as $\boldsymbol{H}_1 = [\boldsymbol{Q}, \boldsymbol{I}]$, where $\boldsymbol{I}$ is an $(n-k) \times (n-k)$ identity matrix. Then permute the corresponding entries of the ratio vector $\boldsymbol{\phi}$ according to above permutation to obtain a new $\boldsymbol{\phi}_1$.
2) **$\boldsymbol{G}_1$.**
   Obtain a new generator matrix $\boldsymbol{G}_1$ from $\boldsymbol{H}_1$. It is easy to observe that $\boldsymbol{G}_1$ can be selected as $[\boldsymbol{I}, -\boldsymbol{Q}^T]$, where $\boldsymbol{I}$ is a $k \times k$ identity matrix and $(\cdot)^T$ denotes the transpose.
3) **$\boldsymbol{G}_2$ and $\boldsymbol{\phi}_2$.**
   Rearrange the columns ($i = 0 \sim n-1$) of $\boldsymbol{G}_1$ according to the descending order of the absolute values of the components in $\boldsymbol{\phi}_1$. Then a new $\boldsymbol{G}_2$ is obtained. Meanwhile, a new ratio vector $\boldsymbol{\phi}_2$ is obtained from $\boldsymbol{\phi}_1$ by the same descending order.
4) **$\boldsymbol{G}^*$ and $\boldsymbol{\phi}^*$.**
   Select k independent columns of $\boldsymbol{G}_2$ from left to right in order, and then append the remaining $n-k$ columns according to the column order of $\boldsymbol{G}_2$. Continue to linearly combine the rows, and obtain a generator matrix $\boldsymbol{G}^*$, whose first k columns form an identity matrix. Meanwhile, a new ratio vector $\boldsymbol{\phi}^*$ is obtained from $\boldsymbol{\phi}_2$ according to the column permutations of $\boldsymbol{G}_2$ in this step.
5) **Binary Tree.**
   According to $\boldsymbol{G}^*$, construct a binary tree with $n+1$ levels, see Fig. 2. For $0 \le i < k$, the number of nodes at the i-th level is 2^i. For $k \le i \le n$, the number of nodes at the i-th level is 2^k. Note that, the code $\boldsymbol{C}^*$ generated by $\boldsymbol{G}^*$ and the code $\boldsymbol{C}$ generated by $\boldsymbol{G}$ are equivalent because of the column permutations and the linear row combinations.

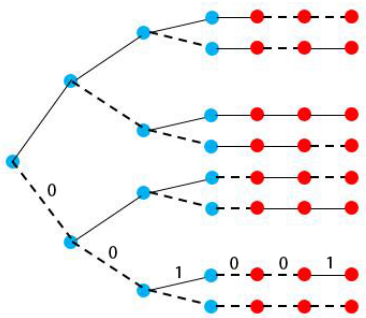


FIGURE 2. The final code tree with labels (0 and 1) represented by dash and solid edges here.

To illustrate clearly, an example is shown in Example 1.

Example 1: This example shows the procedures of constructing a code tree in Fig. 2 for a linear block code $\boldsymbol{C}$ with

generator matrix $\boldsymbol{G}$ and parity check matrix $\boldsymbol{H}$ where

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

Suppose $\boldsymbol{\phi} = (-1.2, -1.0, 0.4, 0.8, -0.9, 0.7)$.

First, exchange the last two columns of $\boldsymbol{H}$ to obtain a new parity check matrix $\boldsymbol{H}_1$ so that the last three columns of $\boldsymbol{H}_1$ form an identity matrix. Then accordingly permute the last two entries of $\boldsymbol{\phi}$ to get a new $\boldsymbol{\phi}_1 = (-1.2, -1.0, 0.4, 0.8, 0.7, -0.9)$. A new generator matrix $\boldsymbol{G}_1$ can be obtained directly from $\boldsymbol{H}_1$, where

$$\boldsymbol{G}_1 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}.$$

Second, order the components of $\boldsymbol{\phi}_1$ and columns of $\boldsymbol{G}_1$ similarly to obtain $\boldsymbol{\phi}_2 = (-1.2, -1.0, -0.9, 0.8, 0.7, 0.4)$ and a new $\boldsymbol{G}_2$ respectively. Then according to the related steps of this subsection, the corresponding ratio vector $\boldsymbol{\phi}^* = (-1.2, -1.0, 0.7, -0.9, 0.8, 0.4)$ and $\boldsymbol{C}^*$ can be obtained respectively when the new generator matrix $\boldsymbol{G}^*$ is determined. In fact,

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

Finally, the code tree is generated in Fig. 2. Since the first three components of all codewords in $\boldsymbol{C}^*$ are message bits, the corresponding layers of code tree should include the following blue nodes in the structure. Meanwhile, the rest layers obtained from $\boldsymbol{G}^*$ lead to the structure including the following red nodes. Note that the dash edge and the solid edge of Fig. 2 represent the bit 0 and bit 1, respectively. In particular, for a message vector $u = (0, 0, 1)$, the corresponding codeword $c = (0, 0, 1, 0, 0, 1) = u \cdot \boldsymbol{G}^* \in \boldsymbol{C}^*$ is illustrated in the following graph.

2) METRIC

To decide which paths are the L (list size) most promising paths in the code tree, a proper metric should be defined. Assume that the root node of the code tree is at level 0. Let $m_{l,i}$ denote the i-th reserved node at level l in the code tree, where $m_{l,i}.left$ and $m_{l,i}.right$ are the left and right child nodes of $m_{l,i}$, respectively. Let t denote the number of reserved paths when the list decoding algorithm reaches the l-th level, such that $t \leq L \leq 2^k$.

Case 1: When $0 \leq l < k$ and $1 \leq i \leq t$, visit the children of each $m_{l,i}$ at the l-th level and define the cost function iteratively

$$\begin{aligned} g(m_{l,i}.left) &= \sum_{j=0}^{l-1} (\phi_j^* - (-1)^{v_{j,i}})^2 + (\phi_l^* - (-1)^0)^2, \\ g(m_{l,i}.right) &= \sum_{j=0}^{l-1} (\phi_j^* - (-1)^{v_{j,i}})^2 + (\phi_l^* - (-1)^1)^2, \end{aligned} \quad (4)$$

where $v_{0,i}, v_{1,i}, \ldots, v_{l-1,i}$ are the labels of edges in the i-th reserved path selected by the list decoding algorithm so far. On the right side of (4), the superscripts 0, 1 in the second term are the labels of the edges between node $m_{l,i}$ and its children. Note that, in above iterative procedure, the t reserved paths of last step branch into $2t$ paths. If $2t \leq L$, the list decoding algorithm reserves the $2t$ paths directly. However, if $2t > L$, the list decoding algorithm prefers the L smallest costs and the corresponding paths.

Case 2: When $k \leq l \leq n$, t should satisfy $t = L$ because of the inequality $L \leq 2^k$ in the list decoding. Currently, it is obvious that $1 \leq i \leq L$. Since each parent node at level l has only one child, the algorithm only needs to follow the L reserved paths in *Case 1*.

At the end of the first phase, only L paths are reserved. Select the one with the smallest cost $g(\cdot)$ among the L paths and output the corresponding label sequence $\boldsymbol{c}_d$. Actually, $\boldsymbol{c}_d$ may not be the optimal codeword, but it can be used to reduce the computational complexity of the second phase.

B. THE SECOND PHASE

After finishing the first phase, the algorithm goes on to the second phase by revisiting the code tree where the A^* algorithm is applied and improved. For the Metric, this subsection modifies two aspects of the heuristic function used in [13], which will be introduced in the **heuristic function** part.

In our improved A^* algorithm, let m denote a node at level l in the code tree, and the estimation function $f(m)$ is composed of actual cost function $g(m)$ and heuristic function $h(m)$. Similar to A^* algorithm, it will overview the list OPEN to find a node in each step with minimum estimation function $f(\cdot)$ for expansion.

1) COST FUNCTION

Let $v_0, v_1, \ldots, v_{l-1}$ denote the labels of edges in the path $\boldsymbol{P}_m$ from the start to node m found by the algorithm. The cost function $g(m)$ is constructed as

$$g(m) = \sum_{j=0}^{l-1} (\phi_j^* - (-1)^{v_j})^2. \quad (5)$$

After constructing the cost function, a heuristic function, which decides the computational complexity of the algorithm directly, should be considered more carefully.

2) HEURISTIC FUNCTION

To construct a more accurate heuristic function h, two properties of linear block codes should be considered.

First, consider the property of code $\boldsymbol{C}^*$. Since the first k symbols of the transmitted codewords in $\boldsymbol{C}^*$ are information symbols, all the entries of $\boldsymbol{v}$, whose orders are larger than k can be obtained if all the first k entries are known. In particular, some latter entries of $\boldsymbol{v}$ can be obtained if parts of the first k entries are known, as shown in Example 2.

Example 2: Let $\boldsymbol{G}^*$ denote the generator matrix of a code $\boldsymbol{C}^*$, which can be expressed as

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

Let $\boldsymbol{u}$ and $\boldsymbol{c}$ denote a message vector and a transmitted codeword respectively, and they satisfy the function: $\boldsymbol{c} = \boldsymbol{u} \times \boldsymbol{G}^*$. For an estimation codeword $\boldsymbol{v}(\boldsymbol{v} \in \boldsymbol{C}^*)$ for $\boldsymbol{c}$, it satisfies $\boldsymbol{v} = \boldsymbol{u'} \times \boldsymbol{G}^*$. In this example, assume $\boldsymbol{u'} = (u'_0, u'_1, u'_2)$, then

$$\begin{aligned} \boldsymbol{v} = \boldsymbol{u'} \times \boldsymbol{G}^* &= \begin{bmatrix} u'_0 & u'_1 & u'_2 \end{bmatrix} \\ &\times \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \\ &= \begin{bmatrix} u'_0 & u'_1 & u'_2 & u'_0 + u'_1 & u'_1 + u'_2 & u'_0 \end{bmatrix} \end{aligned} \quad (6)$$

Assume the estimation codeword $\boldsymbol{v} = (v_0, v_1, v_2, v_3, v_4, v_5)$, which means $v_0 = u'_0$,$v_1 = u'_1$,$v_2 = u'_2$,$v_3 = u'_0 + u'_1 = v_0 + v_1$,$v_4 = v_1 + v_2$,$v_5 = v_0$. Thus, if v_0 and v_1 are known, v_3 can be decided by using v_0, v_1 and $\boldsymbol{G}^*$. And v_4 is only decided by v_1, v_2 and $\boldsymbol{G}^*$, while v_5 is only decided by v_0 and $\boldsymbol{G}^*$

By using these properties, the heuristic function h can be constructed as follows:

1) For nodes at level l, $0 \le l < k$:

Let m denote a node at level l and $v_0, v_1, \ldots, v_{l-1}$ be the labels of edges in the path $\boldsymbol{P}_m$ from the start node to m selected by the algorithm so far. Furthermore, let $HW_{\boldsymbol{C}^*}$ and $d_H(x, y)$ denote the set of all distinct Hamming weights that codewords of $\boldsymbol{C}^*$ may have and the Hamming distance between x and y, respectively. Then we construct a set $S(m)$ for all binary n-tuples $\boldsymbol{v}$ taking advantage of four properties. Actually, by using the set $S(m)$ in heuristic function, our improved A^* algorithm can use more information of partial decoded symbols to improve the accuracy of the function and take advantage of Hamming weight and Hamming distance to reduce search space. The first two properties are about partial decoded symbols, and the third and fourth properties are about Hamming weight and Hamming distance, see follows.

- **Known entries:** the first l entries are the labels of $\boldsymbol{P}_m$;
- **Using partial decoded symbols:** the p-th entry $\breve{v}_p$ is decided by the first l entries and the generator matrix $\boldsymbol{G}^*$, and Example 2 explains the details of this property;
- **Inclusion relationship of Hamming weight:** considering $\boldsymbol{v} \in \boldsymbol{C}^*$, it's easy to conclude that the Hamming weight of $\boldsymbol{v}$ is included in the set of Hamming weight of $\boldsymbol{C}^*$, that is, $HW(\boldsymbol{v}) \in HW_{\boldsymbol{C}^*}$;
- **Inclusion relationship of Hamming distance:** considering $\boldsymbol{v}, \boldsymbol{c}_d \in \boldsymbol{C}^*$, it's easy to conclude that the Hamming distance of $\boldsymbol{v}$ and $\boldsymbol{c}_d$ is included in the set of Hamming weight of $\boldsymbol{C}^*$, that is, $d_H(\boldsymbol{v}, \boldsymbol{c}_d) \in HW_{\boldsymbol{C}^*}$.

All these properties can be realized by using

$$\begin{aligned} &S(m) \\ &= \{\boldsymbol{v} | \boldsymbol{v} = (v_0, \ldots, v_{l-1}, v_l, \ldots, v_{p-1}, \breve{v}_p, \\ &\quad v_{p+1}, \ldots, v_{n-1}), HW(\boldsymbol{v}) \in HW_{\boldsymbol{C}^*}, d_H(\boldsymbol{v}, \boldsymbol{c}_d) \in HW_{\boldsymbol{C}^*}\}. \end{aligned}$$

The heuristic function is

$$h(m) = \min_{\boldsymbol{v} \in S(m)} \sum_{j=l}^{n-1} (\phi_j^* - (-1)^{v_j})^2. \quad (7)$$

Actually, when the second phase algorithm obtains the minimum value in (7), it will calculate $f(m) = g(m) + h(m)$ for the next step of the (improved) A^* algorithm, i.e., it will overview the list OPEN to find a node with minimum estimation function $f(\cdot)$ for expansion.

It should be pointed out that, (1) the vectors in $S(m)$ are selected from the vector space, rather than the code space $\boldsymbol{C}^*$. The code space may not be used since the computational complexity of h is exponential if the vectors are selected from it, while it is $\mathcal{O}(n)$ if the vectors are selected from the vector space, which is presented in [13]; (2) the number of the entries which are decided by the first l entries and the generator matrix $\boldsymbol{G}^*$ may be more than one. Since the positions of these entries may not be interpreted clearly, we just use one symbol $\breve{v}_p$ to represent.

2) For nodes at level l, $k \le l \le n$:

Because the first k columns of $\boldsymbol{G}^*$ are linearly independent, there is only one path from any node at level l to the goal node. Furthermore, it is easy to determine $v_l^*, v_{l+1}^*, \ldots, v_{n-1}^*$ of this path by using $\boldsymbol{G}^*$. The function h is defined as

$$h(m) = \sum_{j=l}^{n-1} (\phi_j^* - (-1)^{v_j^*})^2, \quad (8)$$

where $v_l^*, v_{l+1}^*, \ldots, v_{n-1}^*$ are the labels of edges in the only path $\boldsymbol{P}_m$ from node m to the goal node.

To guarantee that when a node is selected for expansion, our A^* algorithm can find a minimum cost path from the start node to a goal node, the heuristic function h should satisfy the following condition:

3) Condition

For all nodes m_i and their immediate successors m_j,

$$h(m_i) \le h(m_j) + c(m_i, m_j), \quad (9)$$

where $c(m_i, m_j)$ is the cost of the path between node m_i and node m_j. And $c(m_i, m_j)$ can be expressed as $(\phi_j^* - (-1)^{v_j})^2$ where v_j is the label of edges in the path between m_i and m_j, and ϕ_j^* is the corresponding entry of ratio vector $\boldsymbol{\phi}^*$.

Lemma 1: Our improved A^* algorithm used in the second phase satisfies the Condition.

Proof: Let m_i denote the node at level l and m_j be an immediate successor of m_i. The proof should consider three cases.

Case 1 ($l < k-1$): Let $\boldsymbol{v} = (v_0, v_1, \ldots, v_{l-1}, v_l, v_{l+1}, \ldots, v_{n-1})$ belong to $S(m_j)$. The vector $\boldsymbol{v}$ satisfies $d_H(\boldsymbol{v}, \boldsymbol{c}_d) \in HW_{\boldsymbol{C}^*}$ and $HW(\boldsymbol{v}) \in HW_{\boldsymbol{C}^*}$,which means that the vector $\boldsymbol{v}$ also belongs to $S(m_i)$. It is easy to obtain the inequality $\min_{\boldsymbol{v} \in S(m_j)} \sum_{p=l+1}^{n-1} (\phi_p^* - (-1)^{v_p})^2 + c(m_i, m_j) \ge \min_{\boldsymbol{v} \in S(m_i)} \sum_{p=l}^{n-1} (\phi_p^* - (-1)^{v_p})^2$ by observing the definition of the function h, i.e., $h(m_i) \le h(m_j) + c(m_i, m_j)$.

Case 2 ($l = k-1$): The level of m_i is $k-1$ and the level of m_j is k. Then $h(m_j)$ is calculated by using (8), while $h(m_i)$ is calculated by using (7). Since $\boldsymbol{v}$ also belongs to $S(m_i)$, $h(m_i) \leq h(m_j) + c(m_i, m_j)$ holds.

Case 3 ($l \geq k$): Since the levels of m_i and m_j are both larger than or equal to k, $h(m_i)$ and $h(m_j)$ are both calculated by using (8), which means $h(m_i) = h(m_j) + c(m_i, m_j)$. □

The above discussion shows that the A^* algorithm used in our algorithm can output a maximum likelihood decoding result.

Theorem 1: The improved A^* algorithm is a maximum likelihood decoding for all linear block codes.

Proof: The improved A^* algorithm has two phases, where the list decoding algorithm is applied in the first phase to obtain a more accurate seed codeword and the A^* algorithm is employed in the second phase to obtain the final decoding result by using a modified heuristic function. It means that the first phase only reduces the computational complexity of the algorithm rather than decides whether the algorithm is a maximum likelihood decoding algorithm. Only the second phase decides it, i.e., the improved A^* algorithm is a maximum likelihood decoding algorithm if the algorithm used in the second phase can output a maximum likelihood decoding result. Since it is right for the A^* algorithm used in our algorithm, the theorem is proved. □

After learning that the improved A^* algorithm is a maximum likelihood decoding for all linear block codes, the next step is to determine an upper bound on the computational complexity of the improved A^* algorithm in the first and second phases.

In this paper, the computational complexity is defined as the number of nodes visited by the algorithm. Let T_1 and T_2 denote the number of nodes visited by the list decoding algorithm and the average number of nodes visited by the improved A^* algorithm, respectively. And T is the total computational complexity of these two phases.

Due to the list decoding algorithm, it is easy to obtain that the computational complexity in the first phase satisfies

$$T_1 \leq 2Lk + L(n-k) = Lk + Ln, \tag{10}$$

where L is the number of reserved paths during the list decoding process. And the number of visited nodes from the start node to layer k is less than $2Lk$. Then, from layer k to a goal node, since there is only one path, the number of visited nodes is $L(n-k)$.

In order to ensure a low computational complexity of the second phase, the following definition is given to restrict an upper bound on the complexity of the improved A^* algorithm.

For a binary $[n, k]$ linear block code, let $\mathcal{N}$ denote a set of codewords and any codeword in $\mathcal{N}$, namely $\boldsymbol{c}$, satisfies $d_E(\boldsymbol{c}, \boldsymbol{r}) \leq d_E(\boldsymbol{c}_d, \boldsymbol{r})$, where $\boldsymbol{c}_d$ is the seed codeword obtained from the first phase and $\boldsymbol{r}$ is the received vector. And $d_E(\boldsymbol{x}, \boldsymbol{y})$ is the Euclidean distance between $\boldsymbol{x}$ and $\boldsymbol{y}$.

Now, an upper bound on the computational complexity of the second phase can be obtained as follows. The proof of the following theorem is omitted since it is similar to [18]. For a binary $[n, k]$ linear block code, n is the length of a codeword. Since the result of Theorem 2 is derived by using the central limit theorem, it only holds when n is large.

Theorem 2: Let $\boldsymbol{N}$ denote the cardinality of $\mathcal{N}$. Furthermore, denote $Q(\cdot)$ as the standard normal distribution and $k = \lceil \log \boldsymbol{N} \rceil$. Then, for a large n

$$T_2 \leq \widetilde{N}, \tag{11}$$

where

$$\widetilde{N} = 2\left[k + \sum_{l=0}^{k-1}\sum_{\overline{d}=1}^{l}\binom{l}{\overline{d}} Q\left(-\frac{\overline{\mu}(l,\overline{d})}{\overline{\sigma}(l,\overline{d})}\right)\right],$$

$$\overline{\mu}(l,\overline{d}) = \sqrt{N_0}\left\{2\overline{d}\sqrt{R\gamma_b} + (n-l) \cdot \left[2\sqrt{R\gamma_b}Q(-\sqrt{2R\gamma_b}) - \frac{1}{\sqrt{\pi}}e^{-R\gamma_b}\right]\right\}, \tag{12}$$

and

$$\overline{\sigma}^2(l,\overline{d}) = N_0\left\{2\overline{d} + (n-l)\left[(4R\gamma_b+2)Q(-\sqrt{2R\gamma_b}) - 2\sqrt{\frac{R\gamma_b}{\pi}}e^{-R\gamma_b} - \left(2\sqrt{R\gamma_b}Q(-\sqrt{2R\gamma_b}) - \frac{1}{\sqrt{\pi}}e^{-R\gamma_b}\right)^2\right]\right\}. \tag{13}$$

N_0 is noise power per hertz. The signal to noise ratio (SNR) for the channel is $\gamma = E/N_0$. The SNR per transmitted information bit is $\gamma_b = \gamma n/k$. $R = k/n$ is the code rate and d is Hamming distance. $Q(-\overline{\mu}(l,d)/\overline{\sigma}(l,\overline{d}))$ is an upper bound on the probability of a node being expanded, where the node is at level l and the sequence of the labels of the path from the start node to this node have Hamming distance $\overline{d}$ to the transmitted codeword.

Obviously, the total computational complexity of our improved A^* algorithm is $T = T_1 + T_2$.

Finally, the computational complexity and decoding accuracy of the improved A^* decoding algorithm can be concluded as follows. In the first phase, the list decoding algorithm is used to obtain a more accurate seed codeword, which can reduce the computational complexity of the first phase to $T_1 (\leq Lk + Ln)$. In the second phase, it is easy to find out that the improved A^* decoding algorithm is a maximum likelihood decoding algorithm since A^* algorithm is used in this phase, which means the output of this phase is a maximum likelihood decoding result. Also, the computational complexity of the second phase is T_2 ($\leq \widetilde{N}$).

IV. SIMULATION RESULTS FOR THE AWGN CHANNEL

This section presents simulation results over the additive white Gaussian noise (AWGN) channel. Assuming 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

$$c_j = (-1)^{c_j}\sqrt{\varepsilon} \text{ and } r_j = (-1)^{c_j}\sqrt{\varepsilon} + e_j, \tag{14}$$

TABLE 1. The average number of nodes visited by various decoding algorithms for $RM(2, 5)$.

γ_b	3 dB	3.5 dB	4 dB	4.5 dB	5 dB
The improved A^* algorithm	1292	1092	957	851	811
A^* algorithm	1661	1536	1412	1328	1279
Viterbi algorithm with the best code permutation	6396	6396	6396	6396	6396
Viterbi algorithm with the worst code permutation	262,140	262,140	262,140	262,140	262,140
Syndrome trellis algorithm	2,097,152	2,097,152	2,097,152	2,097,152	2,097,152
Exhaustive search(ES) algorithm	2,097,152	2,097,152	2,097,152	2,097,152	2,097,152

TABLE 2. The average number of nodes visited by decoding algorithms for [128, 64] binary extended BCH code.

γ_b	1 dB	1.25 dB	1.5 dB	1.75 dB	2 dB
The improved A^* algorithm	52473	46937	40486	32681	28625
Suboptimal A^* algorithm	59273	53826	47281	40231	35692
A^* algorithm	88325	82650	75905	65223	55474
Exhaustive search algorithm	3.69e19	3.69e19	3.69e19	3.69e19	3.69e19

where ε is the signal energy per channel bit and e_j is a noise sample of a Gaussian process with single side noise power per hertz. The variance of e_j is $N_0/2$.

The Reed-Muller code is a classical error correcting code. Owing to the low complexity of coding and decoding algorithms, the Reed-Muller code can be easily implemented by hardware. In addition, by modifying parameters, the Reed-Muller code can form many different subclasses, which can be adapted to the transmission over different channels. Since the Reed-Muller code was introduced by Muller and Reed in 1954, it has been used in many communication systems, such as deep space communication systems, cellular communication systems and so on.

Consider the r-th order binary Reed-Muller code, $RM(r, m)$, which is an $[n, k]$ linear block code with $n = 2^m$ and $k = 1 + \sum_{i=1}^{r} \binom{n}{m}$. In our simulation, $\boldsymbol{C}$ is $RM(2, 5)$ where $n = 32$, $k = 16$.

Besides, for long block code such as the [128, 64] binary extended BCH code, there is another simulation to compare the performance of different algorithms for it.

A. COMPUTATIONAL COMPLEXITY

For the comparison, the A^* algorithm can use the information contained in the parity bits to reduce the number of search tree edges [15]. TABLE 1 gives the computational complexity of the A^* algorithm for $RM(2, 5)$. The Viterbi algorithm to a minimal trellis and a syndrome trellis of the code are considered. The computational complexity of Viterbi decoding in the minimum trellis was measured in [20]. However, different permutations of a code can have different minimal trellises with different computational complexities. This subsection considers the computational complexities of the best code permutation and the worst code permutation. Due to the difficulty of finding the minimum trellis, [9] applied the Viterbi algorithm to the syndrome trellis of a code. Since the main idea of syndrome trellis algorithm is exhaustive search, it has the same computational complexity with Exhaustive search(ES) algorithm. And ES algorithm can obtain a decoding result by traversing the whole codeword space, which means ES algorithm has the highest computational complexity in all decoding algorithms.

TABLE 2 compares the performance of the improved A^* algorithm, suboptimal A^* algorithm (threshold = 0.25) [18], A^* algorithm and exhaustive search(ES) algorithm for the [128, 64] binary extended BCH code. Suboptimal A^* algorithm uses priority-first search strategy, which reduces the decoding search space and results in an efficient optimal soft-decision decoding algorithm for linear block codes. As shown in the following part, we also compare the bit error probability of improved A^* algorithm with that of adaptive belief propagation (ABP) algorithm. ABP algorithm has been proposed in [21] wherein the graph of the code is adaptively modified at each iteration so that the bit reliabilities can converge in the direction of the correct codeword. Actually, ABP algorithm has much lower complexity than the maximum likelihood soft decoding algorithms. However, since it is an iterative decoding algorithm with different complexity metric, we do not show its computational complexity in TABLE 2. In contrast, its bit error probability is shown in Fig. 4.

TABLE 1 and TABLE 2 show the computational complexity for $RM(2, 5)$ and [128, 64] binary extended BCH code respectively, which is defined as the number of nodes visited by each decoding algorithm. From TABLE 1 and TABLE 2, it is easy to conclude that the improved A^* algorithm has lower computational complexity than other algorithms for the Reed-Muller code and the extended BCH code.

B. BIT ERROR PROBABILITY

As shown in Subsection IV-A, the computational complexity of the improved A^* algorithm is lower than others in TABLE 1 and TABLE 2. Furthermore, it has been proved that the algorithm follows the maximum likelihood decoding rule. The bit error probability of $RM(2, 5)$ is shown in Fig. 3. For comparison, this subsection uses the ES algorithm to decode the received vector $\boldsymbol{r}$ by searching the codeword with the minimum Euclidean distance to the received vector $\boldsymbol{r}$.

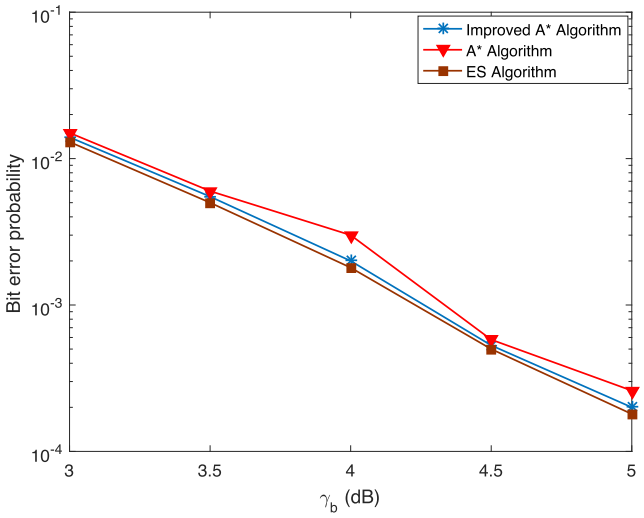


FIGURE 3. The bit error probabilities of various decoding algorithms when the signal-to-noise ratio per information bit of $RM(2, 5)$ is between 3 and 5 dB.

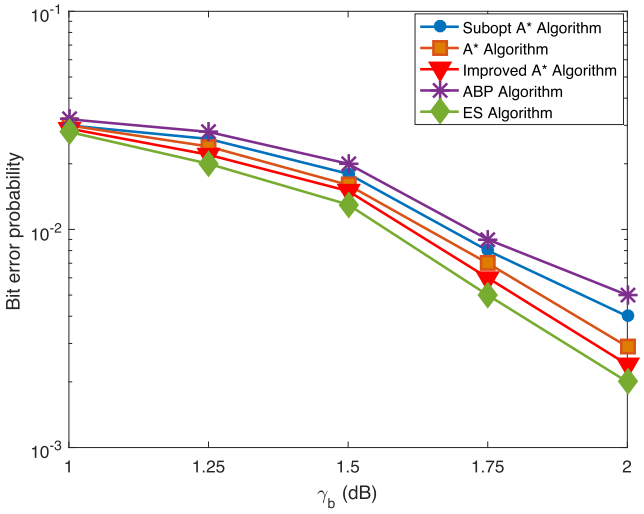


FIGURE 4. The bit error probabilities of various decoding algorithms when the signal-to-noise ratio per information bit of [128, 64] binary extended BCH code is between 1 and 2 dB.

Maximum likelihood decoding algorithm can obtain a decoding result which satisfies the maximum likelihood principle. And ES algorithm can obtain a decoding result by traversing the whole codeword space where the result can minimize the bit error probability. However, in all decoding algorithms, ES algorithm has the highest computational complexity. In particular, the computational complexity of it increases exponentially with the increasing length of a code.

Fig. 3 shows the bit error probabilities of the improved A^* algorithm, A^* algorithm and the ES algorithm for $RM(2, 5)$. In this figure, the error probabilities of the improved A^* algorithm and the ES algorithm are almost the same, which is less than A^* algorithm.

And Fig. 4 shows the bit error probabilities of five decoding algorithms for extended BCH code. In addition to the algorithms based on A^* and ES algorithm, there is an iterative algorithm, namely ABP algorithm, which is shown in Fig. 4. And in this figure, the bit error probability of the improved A^* algorithm is also the closest to the ES algorithm. The comparison results of Fig. 3 and Fig. 4 confirm that the new decoding algorithm reduces the computational complexity without reducing the decoding accuracy.

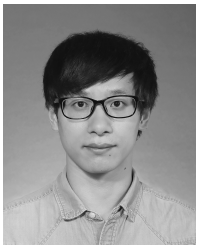
V. CONCLUSION

This paper proposes an improved A^* decoding scheme for linear block codes. The novel decoding algorithm has two phases, where the list decoding algorithm is applied in the first phase and the A^* algorithm is improved and applied in the second phase. The theoretical and simulation works show that the computational complexity of the proposed algorithm is much lower than other decoding algorithms while maintaining decoding accuracy. Future work will be on the decoding of more linear block codes.

REFERENCES

[1] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*. Amsterdam, The Netherlands: Elsevier, 1977.
[2] E. R. Berlekamp, "The technology of error-correcting codes," *Proc. IEEE*, vol. 68, no. 5, pp. 564–593, May 1980.
[3] J. Conway and N. Sloane, "Soft decoding techniques for codes and lattices, including the Golay code and the Leech lattice," *IEEE Trans. Inf. Theory*, vol. IT-32, no. 1, pp. 41–50, Jan. 1986.
[4] Y. S. Han, "A new treatment of priority-first search maximum-likelihood soft-decision decoding of linear block codes," *IEEE Trans. Inf. Theory*, vol. 44, no. 7, pp. 3091–3096, Nov. 1998.
[5] I. Sason and S. Shamai, *Performance Analysis of Linear Codes Under Maximum-Likelihood Decoding: A Tutorial*. Breda, The Netherlands: Now, 2006.
[6] Y. S. Han, T.-Y. Wu, P.-N. Chen, and P. K. Varshney, "A low-complexity maximum-likelihood decoder for tail-biting convolutional codes," *IEEE Trans. Commun.*, vol. 66, no. 5, pp. 1859–1870, May 2018.
[7] V. Guruswami, L. Jin, and C. Xing, "Efficiently list-decodable punctured Reed-Muller codes," *IEEE Trans. Inf. Theory*, vol. 63, no. 7, pp. 4317–4324, Jul. 2017.
[8] V. Guruswami and C. Xing, "List decoding Reed-Solomon, algebraic-geometric, and Gabidulin subcodes up to the Singleton bound," in *Proc. 45th ACM Symp. Theory Comput.*, Palo Alto, CA, USA, May 2013, pp. 843–852.
[9] J. Wolf, "Efficient maximum likelihood decoding of linear block codes using a trellis," *IEEE Trans. Inf. Theory*, vol. IT-24, no. 1, pp. 76–80, Jan. 1978.
[10] R. W. D. Booth, M. A. Herro, and G. Solomon, "Convolutional coding techniques for certain quadratic residual codes," in *Proc. Int. Telemetering Conf.*, Silver Spring, MD, USA, Oct. 1975, pp. 168–177.
[11] H. Gluesing-Luerssen and G. D. Forney, "Reducing complexity of tail-biting trellises," in *Proc. IEEE Int. Symp. Inf. Theory*, Cambridge, MA, USA, Jul. 2012, pp. 646–650.
[12] N. J. Nilsson, *Principles of Artificial Intelligence*. Wellsboro, PA, USA: Tioga Publishing Company, 1980.
[13] Y. S. Han, C. R. P. Hartmann, and C.-C. Chen, "Efficient priority-first search maximum-likelihood soft-decision decoding of linear block codes," *IEEE Trans. Inf. Theory*, vol. 39, no. 5, pp. 1514–1523, Sep. 1993.
[14] L. Ekroot and S. Dolinar, "A^* decoding of block codes," *IEEE Trans. Commun.*, vol. 44, no. 9, pp. 1052–1056, Sep. 1996.
[15] T.-H. Chen, K.-C. Chen, M.-C. Lin, and C.-F. Chang, "On A^* algorithms for decoding short linear block codes," *IEEE Trans. Commun.*, vol. 63, no. 10, pp. 3471–3481, Oct. 2015.
[16] J. Ortín, P. García, F. Gutiérrez, and A. Valdovinos, "A^* based algorithm for reduced complexity ML decoding of tailbiting codes," *IEEE Commun. Lett.*, vol. 14, no. 9, pp. 854–856, Sep. 2010.
[17] A. Barg, E. Krouk, and H. C. A. V. Tilborg, "On the complexity of minimum distance decoding of long linear codes," *IEEE Trans. Inf. Theory*, vol. 45, no. 5, pp. 1392–1405, Jul. 1999.

[18] Y. S. Han, C. R. P. Hartmann, and K. G. Mehrotra, "Decoding linear block codes using a priority-first search: Performance analysis and suboptimal version," *IEEE Trans. Inf. Theory*, vol. 44, no. 3, pp. 1233–1246, May 1998.
[19] T.-Y. Hwang, "Decoding linear block codes for minimizing word error rate," *IEEE Trans. Inf. Theory*, vol. IT-25, no. 6, pp. 733–737, Nov. 1979.
[20] R. J. McEliece, "The Viterbi decoding complexity of linear block codes," presented at the IEEE Int. Symp. Inf. Theory, Trondheim, Norway, Jun. 1994.
[21] J. Jiang and K. R. Narayanan, "Iterative soft decision decoding of reed-Solomon codes based on adaptive parity check matrices," presented at the IEEE Int. Symp. Inf. Theory, Chicago, IL, USA, Jun. 2004.



BIN XU received the B.S. degree in ocean engineering from Shanghai Jiao Tong University, China, in 2016. He is currently pursuing the M.S. degree with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His current research includes communication coding algorithm, large-scale data processing, and machine learning.

CHENHAO YING received the B.S. degree in communication engineering from Xidian University, China, in 2016. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His current research includes communication coding algorithms and wireless communications.

YUAN LUO received the B.S. degree in information science and the M.S. and Ph.D. degrees in probability and mathematical statistics from Nankai University, Tianjin, China, in 1993, 1996, and 1999, respectively. From 1999 to 2001, he held a post-doctoral position with the Institute of Systems Science, Chinese Academy of Sciences. From 2001 to 2003, he held another post-doctoral position with the Institute for Experimental Mathematics, University of Duisburg-Essen, Germany. Since 2003, he has been with the Computer Science and Engineering Department, Shanghai Jiao Tong University, Shanghai, China. He has been a Full Professor since 2006. His research interests include information theory, communication coding theory, and computer security.