LETTER

A Call-by-Need Recursive Algorithm for the LogMAP Decoding of a Binary Linear Block Code

Toshiyuki ISHIDA[†a)], ***Student Member and*** **Yuichi KAJI**[†], ***Regular Member***

SUMMARY A new algorithm for the LogMAP decoding of linear block codes is considered. The decoding complexity is evaluated analytically and by computer simulation. The proposed algorithm is an improvement of the recursive LogMAP algorithm proposed by the authors. The recursive LogMAP algorithm is more efficient than the BCJR algorithm for low-rate codes, but the complexity grows considerably large for high-rate codes. The aim of the proposed algorithm is to solve the complexity explosion of the recursive LogMAP algorithm for high-rate codes. The proposed algorithm is more efficient than the BCJR algorithm for well-known linear block codes.

key words: *LogMAP decoding, linear block codes, BCJR algorithm, trellis diagram, turbo codes*

1. Introduction

This letter is to investigate an efficient algorithm for the LogMAP decoding of linear block codes. The LogMAP decoding plays essential roles in many applications of the coding theory. A well-known approach for the LogMAP decoding is to use a trellis-based BCJR algorithm [1]. However, since linear block codes usually have large and complicated trellis diagrams, the complexity of the BCJR algorithm is especially large for linear block codes.

In [5], the authors have proposed a recursive algorithm for the LogMAP decoding. The recursive algorithm is much more efficient than the BCJR algorithm if the code has rather small code rate, while its complexity is worse than the BCJR algorithm if the code rate is not small. The authors analyzed the behavior of the algorithm in detail, and found that a major part of intermediate results, which are obtained in a recursive computation, do not contribute at all to the computation of the upper-level of the recursion. If we can omit the computation of such useless intermediate results, then we can reduce the complexity of the algorithm without affecting the decoding result.

In this letter, we consider to apply the "call-by-need" approach to the recursive LogMAP algorithm, where the "call-by-need" approach is known to be effective to reduce the decoding complexity of the recursive maximum likelihood decoding algorithm [6]. We present a modified recursive LogMAP algorithm, and investigate a non-trivial upper-bound of the space complexity. Unfortunately, the authors cannot present any theoretical results on the time complexity of the algorithm, because the behavior of the algorithm changes drastically and complicatedly for its inputs (a received sequence). Even though, simulation results show that the proposed algorithm is more efficient than the BCJR algorithm even if the code has rather high code rate.

Manuscript received April 28, 2003.
Final manuscript received August 14, 2003.

[†]The authors are with the Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma-shi, 630-0192 Japan.
a) E-mail: toshi-i@is.aist-nara.ac.jp

2. Preliminary

Notations used in this letter are briefly reviewed. Refer [2], [5] for more in detail. The followings are necessary to consider recursive decoding algorithms. For an (n, k) linear block code C, let $p_{x,y}(C) = \{(v_{x+1}, \ldots, v_y) : (v_1, \ldots, v_n) \in C\}$. Also let $C_{x,y} = \{(v_{x+1}, \ldots, v_y) : (v_1, \ldots, v_n) \in C, v_j = 0 \text{ for } j \leq x \text{ and } j > y\}$. It is known that $C_{x,y}$ is a linear subcode of $p_{x,y}(C)$ [2], and $p_{x,y}(C)$ is divided into cosets of $C_{x,y}$. Let $L_{x,y}$ be the set of cosets of $C_{x,y}$ in $p_{x,y}(C)$. It is also known that, for an integer z with $x < z < y$, any coset $D \in L_{x,y}$ is represented as

$$D = \bigcup_{i=1}^{2^q} (D_{L,i} \circ D_{R,i}) \tag{1}$$

where $D_{L,1}, \ldots, D_{L,2^q} \in L_{x,z}$, $D_{R,1}, \ldots, D_{R,2^q} \in L_{z,y}$, "$\circ$" denotes element-wise concatenation, and $q = k(C_{x,y}) - k(C_{x,z}) - k(C_{z,y})$ with $k(\cdot)$ denoting the dimension [2].

Next we redefine the LogMAP decoding. For a received sequence and given *a priori* values of symbols, a *bit metric* $m_i(b)$ is defined for $b \in \{0, 1\}$ and $1 \leq i \leq n$. Roughly speaking, bit metrics correspond to $\log \gamma(\cdot)$ in [4]. For a vector $\boldsymbol{v} = (v_1, \ldots, v_{y-x}) \in p_{x,y}(C)$, define and denote the *metric* of $\boldsymbol{v}$ as $m(\boldsymbol{v}) = \sum_{i=1}^{y-x} m_{x+i}(v_i)$. Consider a coset $D \in L_{x,y}$. For $1 \leq i \leq y - x$ and $b \in \{0, 1\}$, let $D^{(i,b)}$ be the set of vectors which belong to D and have the symbol b at the i-th symbol position, and let $\boldsymbol{v}^{(i,b)}$ be the vector which has the largest metric in $D^{(i,b)}$. Arrange vectors in the set $\{\boldsymbol{v}^{(i,b)} : 1 \leq i \leq y - x, b \in \{0, 1\}\}$ in the decreasing order of metrics, and we can obtain an ordered list of vectors. Let $CVL(D)$ denote this ordered list of vectors, and call the list a *covering vector list* of D. In this letter, we define that the LogMAP decoding for

a coset D is to compute $CVL(D)$. Remark that the code C is a special coset in $L_{0,n}$ and the above discussion also applies to the code C. The LogMAP decoding for C defined as above is essentially equivalent to the conventional definition of the LogMAP decoding [4].

3. Proposed Algorithm

In the recursive LogMAP decoding considered in [5], the LogMAP decoding for the code C is realized by applying an algorithm for computing $CVL(D)$ in a recursive manner. The following lemma is essential for the recursive computation.

Lemma 1: Let z be an integer with $x < z < y$. For any coset $D \in L_{x,y}$ and any vector $\boldsymbol{v} \in CVL(D)$, there exist cosets $D_L \in L_{x,z}$ and $D_R \in L_{z,y}$, and vectors $\boldsymbol{l} \in CVL(D_L)$ and $\boldsymbol{r} \in CVL(D_R)$ such that $\boldsymbol{v} = \boldsymbol{l} \circ \boldsymbol{r}$. □

To follow a "call-by-need" approach, it is convenient to consider a union of covering vector lists. Now define

$$CVL_{x,y} = \bigcup_{D \in L_{x,y}} CVL(D),$$

where the union of lists is defined in the same manner as the union of sets except that vectors in the resultant list are arranged in the decreasing order of metrics. If $y-x$ is small, then $CVL_{x,y}$ is efficiently computable directly in a straight-forward manner. However, if $y-x$ is not small, then such a direct computation of $CVL_{x,y}$ will consume large complexity. Thus we use the following corollary which is a counterpart of Lemma 1.

Corollary 2: For any $\boldsymbol{v} \in CVL_{x,y}$, there exist $\boldsymbol{l} \in CVL_{x,z}$ and $\boldsymbol{r} \in CVL_{z,y}$ such that $\boldsymbol{v} = \boldsymbol{l} \circ \boldsymbol{r}$. □

In the following, we consider an algorithm which finds a vector in $CVL_{x,y}$ one-by-one. Assume that the first $h-1$ vectors in $CVL_{x,y}$ have been correctly computed as $\Lambda = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{h-1}]$ where $h \geq 1$. The following Algorithm 3 computes the vector $\boldsymbol{v}_h$ which occupies the next position to $\boldsymbol{v}_{h-1}$ in $CVL_{x,y}$. Remark that the metric of $\boldsymbol{v}_h$ must be smaller than the metrics of $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_{h-1}$ since vectors in $CVL_{x,y}$ are in the decreasing order of metrics. To make the description simpler, the i-th vectors in $CVL_{x,z}$ and $CVL_{z,y}$ are denoted by $\boldsymbol{l}_i$ and $\boldsymbol{r}_i$, respectively. Vectors $\boldsymbol{l}_i$ and $\boldsymbol{r}_i$ are computable by applying Algorithm 3 to the sections x–z and z–y, respectively. For each $\boldsymbol{v}_m$ with $1 \leq m \leq h-1$, let $i(m)$ and $j(m)$ be integers such that $\boldsymbol{v}_m = \boldsymbol{l}_{i(m)} \circ \boldsymbol{r}_{j(m)}$. Corollary 2 guarantees that such integers $i(m)$ and $j(m)$ uniquely exist. For pairs of integers, we write $(i,j) \geq (i',j')$ if $i \geq i'$ and $j \geq j'$. If $(i,j) \geq (i',j')$ and $(i,j) \neq (i',j')$, then $(i,j) > (i',j')$. For the list Λ and a vector $\boldsymbol{u} \in p_{x,y}(C)$, we write $\Lambda|_{\boldsymbol{u}}$ for the list $[\boldsymbol{v} : \boldsymbol{v} \in \Lambda, \boldsymbol{u}$ and $\boldsymbol{v}$ belong to the same coset in $L_{x,y}]$. We say that a vector $\boldsymbol{u} = (u_1, \ldots, u_{y-x}) \in p_{x,y}(C)$ is *covered* by $\Lambda|_{\boldsymbol{u}}$ if and only if, for any i with $1 \leq i \leq y-x$, there exists $\boldsymbol{v} = (v_1, \ldots, v_{y-x}) \in \Lambda|_{\boldsymbol{u}}$ such that $u_i = v_i$.

Algorithm 3:

1. Let $S := N \times N$ where N is the set of natural numbers.
2. Let $S_h := S \setminus \bigcup_{m=1}^{h-1} \{(i,j) : (i,j) < (i(m), j(m))\}$.
3. Let Γ_h be the set of all pairs of integers $(i,j) \in S_h$ which satisfy the following conditions;
   a. $\boldsymbol{l}_i \circ \boldsymbol{r}_j \in p_{x,y}(C)$,
   b. $\boldsymbol{l}_i \circ \boldsymbol{r}_j$ is not covered by $\Lambda|_{\boldsymbol{l}_i \circ \boldsymbol{r}_j}$, and
   c. the above two conditions do not hold for any $(i',j') \in S_h$ with $(i',j') < (i,j)$.
4. Find a pair $(i,j) \in \Gamma_h$ which maximizes $m(\boldsymbol{l}_i \circ \boldsymbol{r}_j)$ among all pairs in Γ_h, and return $\boldsymbol{l}_i \circ \boldsymbol{r}_j$.

□

The conditions 3(a) and 3(b) are easily possible by using the parity check matrices of $p_{x,y}(C)$ and $C_{x,y}$, respectively [6].

Theorem 4: The vector $\boldsymbol{l}_i \circ \boldsymbol{r}_j$ returned by Algorithm 3 is $\boldsymbol{v}_h$.

Proof. By Corollary 2, there are integers $i(h)$ and $j(h)$ such that $\boldsymbol{v}_h = \boldsymbol{l}_{i(h)} \circ \boldsymbol{r}_{j(h)}$. It suffices to show that

A. $(i(h), j(h))$ belongs to S_h after Step 2,
B. $(i(h), j(h))$ is included in Γ_h at Step 3, and
C. $(i(h), j(h))$ is selected at Step 4.

To show A, we use the property that CVLs are arranged in the decreasing order of metrics. If there is m such that $1 \leq m \leq h-1$ and $(i(h), j(h)) < (i(m), j(m))$, then the metric of $\boldsymbol{v}_h = \boldsymbol{l}_{i(h)} \circ \boldsymbol{r}_{j(h)}$ is larger than that of $\boldsymbol{v}_m = \boldsymbol{l}_{i(m)} \circ \boldsymbol{r}_{j(m)}$. On the other hand, $\boldsymbol{v}_m$ must have the larger metric than $\boldsymbol{v}_h$ since $m < h$, a contradiction. Thus $(i(h), j(h))$ remains in S_h after Step 2.

To prove B, we need to show that $i(h)$ and $j(h)$ satisfy the three conditions (a), (b) and (c) in Step 3, where the condition (a) holds obviously. For the contradiction, assume that (b) does not hold, that is, assume that $\Lambda|_{\boldsymbol{l}_{i(h)} \circ \boldsymbol{r}_{j(h)}}$ covers $\boldsymbol{l}_{i(h)} \circ \boldsymbol{r}_{j(h)}$. In this case, for every $1 \leq i \leq y-x$, there exists $\boldsymbol{v}_m \in \Lambda|_{\boldsymbol{l}_{i(h)} \circ \boldsymbol{r}_{j(h)}}$ which has the same symbol as $\boldsymbol{v}_h$ at the i-th position. Since $\boldsymbol{v}_m$ has the larger metric than $\boldsymbol{v}_h$, this means that $\boldsymbol{v}_h$ cannot have the largest metric in $D^{(i,b)}$ where $D \in L_{x,y}$ is the coset to which $\boldsymbol{v}_h$ belongs and b is the i-th symbol of $\boldsymbol{v}_h$. Therefore, $\boldsymbol{v}_h$ cannot belong to $CVL(D)$ nor $CVL_{x,y}$, a contradiction since we let $\boldsymbol{v}_h$ be a vector in $CVL_{x,y}$. A similar proof by contradiction is possible to show that $i(h)$ and $j(h)$ pass the condition (c) in Step 3. In total, $(i(h), j(h))$ is included in Γ_h.

We omit the discussion on the condition C since it is also shown by a similar proof by contradiction. □

Vectors $\boldsymbol{l}_i$ and $\boldsymbol{r}_i$ are computed by applying Algorithm 3 to the sections x–z and z–y, respectively, when

the vectors are needed for the first time. Vectors which are not referenced are never computed. With this "call-by-need" computation, we can avoid computing useless vectors in CVLs. By applying the algorithm for the section 0–n iteratively, we can compute $CVL_{0,n}$.

4. Decoding Complexity

The complexity of the proposed algorithm is considered. The algorithm has a recursive structure for computing $CVL_{x,y}$. If $y-x$ is small, then the computation is performed directly, while if $y-x$ is not small, then the computation is performed recursively by dividing the section x–y at a pivot z with $x < z < y$. The complexity of the algorithm strongly depends on the sectionalization, that is, the choice of the pivot z in the recursive procedure, and the criteria for switching to the direct computation.

We first consider the upper-bound limit of the space complexity of the algorithm, and present an actual complexity obtained by computer simulation. The upper-bound can be used to look for a good sectionalization which reduces the decoding complexity. For the measure of the space complexity, we consider the number of vectors which must be memorized for a decoding. We omit the proof due to the space limitation, but the following is derived by using a similar discussion for deriving the Grismer bound [3].

Lemma 5: Let D be a coset in $L_{x,y}$. If $|C_{x,y}| = 1$, then $|CVL(D)| = 1$. Otherwise,

$$\begin{aligned} |CVL(D)| &\le 2 + \log_2(d_{x,y}/(2d_{x,y} - (y-x))) \\ &\quad (\text{if } y - x < 2d_{x,y}), \\ |CVL(D)| &\le (y-x) + \log_2 d_{x,y} - 2d_{x,y} + 3 \\ &\quad (\text{otherwise}). \end{aligned}$$

where $d_{x,y}$ is the minimum distance of $C_{x,y}$. □

Let $\phi_{x,y}$ be the upper-bound limit on the space complexity for computing $CVL_{x,y}$, and let $b_{x,y}$ be the upper-bound limit of $CVL(D)$ given by Lemma 5. To make discussion simpler, assume that the direct computation is employed only if $y-x = 1$, and the recursive computation (Algorithm 3) is used if $y - x > 1$. Then, we have

$$\begin{aligned} \phi_{x,y} &= |p_{x,y}(C)| \qquad (\text{if } y - x = 1), \\ \phi_{x,y} &= |L_{x,y}| \times b_{x,y} + \min_{x<z<y} \{\phi_{x,z} + \phi_{z,y}\} \\ &\quad (\text{otherwise}). \end{aligned}$$

By using the above expressions and a dynamic programming technique, we can find a space-optimum sectionalization which minimizes $\phi_{x,y}$ in the cubic order to the section length $y - x$.

Table 1 is to compare the space complexities of the proposed algorithm (the upper-bound and experiment values obtained by computer simulation), recursive LogMAP algorithm (by computer simulation) and the BCJR algorithm (computed from the size of the trellis). RM and exBCH stand for the Reed-Muller and extended and permuted BCH codes [2], respectively. For the computer simulation for the proposed algorithm, the space-optimum sectionalization considered above is used, while for the recursive LogMAP algorithm, a heuristically good sectionalization [5] is used.

Next, we investigate the time complexity of the algorithm. The measure of the time complexity is the number of addition equivalent operations of metrics, which is commonly used to measure the complexity of the maximum likelihood decoding algorithms. Unfortunately, we have not found a non-trivial bound on the time complexity since the behavior of the algorithm changes dynamically, but found by computer simulation that the complexity is much smaller than that of the BCJR algorithm. Table 2 shows the time complexities of the considered LogMAP algorithms.

We can see that the proposed algorithm is more efficient than the BCJR algorithm for all the codes presented in Tables 1 and 2. The decoding complexity of the proposed algorithm changes according to inputs to the algorithm. If the signal-to-noise ratio is large, then one vector in $p_{x,y}(C)$ (which is usually a part of the transmitted vector) has very large metric compared to other vectors, while other vectors have "nearly equally" small metrics. Hence we can find the first vector in a covering vector list efficiently, but we need much complexity to find the second and later vectors in the list. In total, the decoding complexity of the proposed algorithm is not monotonical to the signal-to-noise ratios of the channel. We considered block codes with length up

Table 1 The space complexity.

code	proposed			recursive LogMAP	BCJR-based
E_b/N_0	2dB	4dB	(upper bound $\phi_{0,n}$)	4dB	
RM(64,22)	16,314	18,787	24,103	20,249	649,720
RM(64,42)	37,853	38,417	72,246	52,748	649,720
RM(64,57)	4,047	3,855	4,733	3,821	5,272
exBCH(64,7)C	769	773	776	391	5,272
exBCH(64,10)C	2,333	2,375	2,415	2,185	37,080
exBCH(64,16)B	41,165	41,866	44,567	50,360	1,331,704
exBCH(64,24)B	36,701	38,542	61,375	59,869	1,876,472
exBCH(64,45)C	126,265	119,359	242,622	1,285,741	836,088
exBCH(64,51)B	57,806	55,930	90,357	604,419	243,192

Table 2 The time complexity.

code	proposed		recursive LogMAP	BCJR-based
E_b/N_0	2dB	4dB	4dB	
RM(64,22)	140,904	165,549	158,742	1,975,454
RM(64,42)	349,404	351,376	1,007,709	3,195,806
RM(64,57)	20,911	19,590	33,398	30,238
exBCH(64,07)C	1,852	1,873	667	13,774
exBCH(64,10)C	11,102	11,366	3,556	97,998
exBCH(64,16)B	382,992	392,470	170,035	3,673,246
exBCH(64,24)B	380,291	402,392	608,300	6,053,022
exBCH(64,45)C	1,428,151	1,334,720	5,500,698	4,414,366
exBCH(64,51)B	577,068	555,215	2,396,110	1,373,086

to 64 since block codes with the larger-scale than these parameters are used quite seldom. Indeed we could not find literatures which considers the LogMAP decoding of linear block codes with length more than 64. The tables include results for codes with the practically largest parameters. Due to the overhead for the call-by-need computation, the complexity of the proposed algorithm is not as good as that of the recursive LogMAP algorithm for low-rate codes, though, the complexity of the proposed algorithm is much more smaller than that of the recursive LogMAP algorithm for medium- to high-rate codes.

5. Conclusion

We investigated a call-by-need algorithm for the LogMAP decoding of linear block codes. Due to the overhead for the call-by-need computation, the algorithm is not as efficient as the recursive LogMAP algorithm for low-rate codes. However, the algorithm succeeded to reduce the large complexity of the recursive LogMAP algorithm for high-rate codes. This is mainly thanks to the call-by-need computation which works effectively to avoid useless computation.

References

[1] L.R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," IEEE Trans. Inf. Theory, vol.20, no.2, pp.284–287, March 1974.

[2] T. Fujiwara, H. Yamamoto, T. Kasami, and S. Lin, "A trellis-based recursive maximum likelihood decoding algorithm for linear codes," IEEE Trans. Inf. Theory, vol.44, no.2, pp.714–729, March 1998.

[3] J.H. Grismer, "A bound for error-correcting codes," IBM J. Res. Dev., vol.4, pp.532–542, 1960.

[4] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," IEEE Trans. Inf. Theory, vol.42, no.2, pp.429–445, March 1996.

[5] Y. Kaji, R. Shibuya, T. Fujiwara, T. Kasami, and S. Lin, "MAP and LogMAP decoding algorithms for linear block codes using a code structure," IEICE Trans. Fundamentals, vol.E83-A, no.10, pp.1884–1890, Oct. 2000.

[6] Y. Kaji, T. Fujiwara, and T. Kasami, "An efficient call-by-need algorithm for the maximum likelihood decoding of a linear code," Proc. 2000 Intl. Symp. on Inf. Theory and Its Applications, pp.335–338, Honolulu, Hawaii, Nov. 2000.