IEICE TRANS. COMMUN., VOL.E94–B, NO.4 APRIL 2011

LETTER

On-the-Fly Maximum-Likelihood Decoding of Raptor Codes over the Binary Erasure Channel

Saejoon KIM[†a)], **Seunghyuk LEE**[†], *Nonmembers*, **Jun HEO**[††], *and* **Jongho NANG**[†], *Members*

SUMMARY In this letter, we propose an efficient on-the-fly algorithm for maximum-likelihood decoding of Raptor codes used over the binary erasure channel. It is shown that our proposed decoding algorithm can reduce the actual elapsed decoding time by more than two-thirds with respect to an optimized conventional maximum-likelihood decoding.

key words: *raptor codes, on-the-fly decoding, maximum-likelihood decoding, binary erasure channel*

1. Introduction

Fountain codes are rateless erasure correcting codes that can generate, on-the-fly, as many encoding symbols as necessary for reliable transmission of data. Raptor codes [4] are a realization of fountain codes that are universally capacity-achieving over the binary erasure channel (BEC). For Raptor codes of finite lengths, it is known that maximum-likelihood (ML) decoding can provide noticeably superior performance to the linear-time message-passing decoding [3]. Furthermore, there exist efficient implementations of ML decoding of Raptor codes so that ML decoding is computationally feasible for finite code lengths [2], [5].

In a conventional ML decoding of Raptor codes, decoding begins only after enough encoding symbols have been received which can necessitate a peak in decoder's processing load at the end of data reception. Thus the decoder's processing of Raptor codes is in direct contrast with the encoder's processing in which encoding symbols are generated on-the-fly. Moreover, conventional decoding can lead to an increase in decoder's processing speed requirements for real-time applications. To this end, we propose an *on-the-fly* ML decoding of Raptor codes in which decoding is processed as encoding symbols are received. It will be shown that our proposed decoding algorithm can reduce the actual decoding time elapsed by more than $\frac{2}{3}$ compared to the optimized conventional ML decoding of [2]. In a related work, on-the-fly decoding of LT codes [3] has recently been presented in [1] where decoding time reduction of less than 30% with respect to a standard Gaussian elimination (GE) was obtained. We note that the ML decoding algorithm of [2], to which our proposed algorithm will be compared, runs considerably faster, i.e., > 100 times, than a standard GE.

Manuscript received July 20, 2010.
Manuscript revised December 3, 2010.

[†]The authors are with the Department of Computer Science and Engineering, Sogang University, Seoul, Korea.
[††]The author is with the School of Electrical Engineering, Korea University, Seoul, Korea.
a) E-mail: saejoon@sogang.ac.kr

DOI: 10.1587/transcom.E94.B.1062

2. ML Decoding of Raptor Codes

We shall consider the Raptor codes defined in the 3GPP standard [5] that give excellent performance when ML decoded, and which consist of a pre-code and an LT code in a serially concatenated fashion. Let us denote the message symbols of length k by $\mathbf{m}$, and the intermediate symbols of length $k+\triangle k$ corresponding to the codeword of the pre-code by $\hat{\mathbf{m}}$. Here, $\triangle k$ is the amount of redundancy added by the pre-code. This $\hat{\mathbf{m}}$ is then encoded by the LT code to a stream of encoding symbols which is then transmitted. Let $\mathbf{c}$ and $\hat{\mathbf{c}}$ represent the received encoding symbols of length n, and the concatenation of $0^{\triangle k}$ and $\mathbf{c}$ where $0^{\triangle k}$ is the all-zero symbol string of length $\triangle k$, respectively. Here, $\hat{\mathbf{m}}$ and $\hat{\mathbf{c}}$ are related by $A\hat{\mathbf{m}}^t = \hat{\mathbf{c}}^t$ where A is an $n + \triangle k$ by $k + \triangle k$ matrix and the superscript t represents the transpose operator. In A, the top $\triangle k$ rows represent the constraints of the pre-code on $\hat{\mathbf{m}}$, and the bottom n rows, each corresponding to a received encoding symbol of $\mathbf{c}$, represent the generator matrix of the LT code. Raptor codes of [5] were designed so that the original message $\mathbf{m}$ can be computed very efficiently given $\hat{\mathbf{m}}$, and therefore ML decoding of Raptor codes essentially reduces to GE of the matrix A. We briefly summarize a time-efficient implementation of it that was introduced in [5] and improved in [2] as our on-the-fly algorithm is a modification of [2].

The time-efficient implementation of [2] and [5] consists of four phases after which the original matrix A is converted into an identity matrix through row/column exchange and row addition. In the first phase, the matrix A is converted into a matrix consisting of left upper identity submatrix, left lower all-zero submatrix, and a right submatrix. To convert the matrix A into this form, another matrix V will be utilized which is formed by the intersection of all but the first i columns and the last u columns, and all but the first i rows of A for nonnegative integers i and u. Initially i and u are both set to 0 and therefore $V = A$. During the course of the first phase, the values of i and u will increase and hence the matrix V will change, and if the phase ends successfully, V will disappear, i.e., $i + u = k + \triangle k$.

In a nutshell, the first phase proceeds as follows [2]. A row with the minimum positive weight, say, r in V is chosen for processing and it is exchanged with the first row of V. Then the columns of V are rearranged such that the column of V of one of the r 1's in the chosen row is exchanged with

the first column of V and the columns of the remaining $r-1$ 1's are exchanged with the last columns of V. All rows of V that have a 1 in the first column of V are exclusive-ORed with the first row of V so that all rows below it have a 0 in the first column of V. The value of i is incremented by 1 and that of u is incremented by $r-1$, and this process is iterated repeatedly until no row of positive weight in V can be found.

The goal of the first phase is to make i as large as possible by processing rows of weight two that are contained in components of maximum size [5] or that have the highest score [2]. We note that a component is roughly defined as the set of rows of weight two in which every row shares a column index of 1's locations with another row in the set. Thus, the number of non-zero columns in this component, called the component size, is at most one more than that of rows in the component. Then, when a row in a component of size s is chosen for processing, it will cause all the other rows in the component to be chosen sequentially so that i is incremented by $s-1$ and u is incremented by 1. The score is defined similarly as the component size, and whenever a row of score s is processed, i is incremented by $s-1$ and u is incremented by 1. By abuse of terminology, we will call a component the set of columns that are processed as the result of processing a row by either scheme. Through using the score [2] as the criteria for row selection this phase can be implemented in linear-time.

Denoting by U_1 and U_2 the first i and the remaining rows of the rightmost u columns, a standard GE is applied to U_2 to convert it into a matrix where the first u rows is the identity matrix in the second phase. If this is successful, then the bottom $n-k$ rows of A are discarded and the resulting matrix is a square matrix of size $k+\triangle k$ with 1's along the diagonal. In the remaining phases three and four, the submatrix U_1 is converted into an all-zero matrix through basic operations with the bottom u rows after which A becomes the identity matrix. The computational bottleneck of the described algorithm is clearly the second phase where cubic-time is required for the standard GE.

3. On-the-Fly ML Decoding of Raptor Codes

On-the-fly decoding allows one to start the decoding process as encoding symbols are received rather than wait until the reception of enough encoding symbols. A goal of it is to distribute the decoding process with respect to the number of encoding symbols received so that once enough encoding symbols are received, the required time for decoding completion can be much shorter than otherwise. Hereafter, we will assume that n encoding symbols are necessary for successful decoding and that decoding begins after q fraction of these encoding symbol are received where q is a free parameter.

To construct an efficient on-the-fly ML decoding of Raptor codes, we shall modify the efficient ML decoding algorithm of [2] described in the previous section which is the currently known fastest ML decoding algorithm for Raptor codes. As this algorithm consists of four phases where the first phase can be processed in linear-time while the second phase requires cubic-time, a natural approach is to process the first phase on-the-fly as much as possible and process the remaining phases once enough encoding symbols are received. As the decoding time required for second through fourth phases depends on the size of u, we wish to make it small. On the other hand, since the size of u is the number of components processed in the first phase, these components need be of large size in order to minimize u's value.

Now consider the process of receiving encoding symbols during which our proposed algorithm proceeds with the first phase as follows. First, if an encoding symbol corresponding to a weight one row is received, it is processed immediately as it will not be helpful not do to otherwise. Likewise, if an encoding symbol corresponding to a row of weight greater than two is received, it is not processed immediately as it will greatly increase the value of u if done otherwise. Hence, it suffices to consider the case when an encoding symbol corresponding to a row of weight two is received hereafter. The goal of our proposed algorithm is to select a row of weight two for processing not too often, as encoding symbols are received, so that the size of its component is not small. However, a row of weight two should be selected for processing often enough so that the decoding process is being well distributed with respect to the reception of encoding symbols. In particular, the size of the component chosen for processing cannot be too big for real-time implementations. To this end, let us define the *progress rate* function, $P(x)$, as

$$P(x) \triangleq \frac{n}{k+\triangle k}\frac{i+u}{x}$$

which represents the fraction of processed columns, i.e., $i+u$, with respect to the number of received encoding symbols x normalized such that $P(n)=1$. Note that $P(qn)=0$ since decoding begins after qn encoding symbols are received. Because the value of $i+u$ is calculated anyway during decoding [2], [5], $P(x)$ can be calculated immediately with little additional computational cost. The progress rate function tells us that if its value is large, then the size of the largest component will likely be small since there are only a few unprocessed columns left. On the other hand, the function implies that if its value is small, then the decoding process is not being well distributed. Thus the progress rate function $P(x)$ provides a guideline for the selection of a weight two row for processing. Specifically, we have observed empirically that the value of $P(x)$ should start small and get increasingly larger as x increases in order for the size of the processed components to remain approximately constant and not large. For this matter, let us define the *criterion* function, $C(x)$, as

$$C(x) \triangleq -\left[\frac{1}{(1-q)}\left(1-\frac{x}{n}\right)\right]^{\frac{1}{r}}+1$$

where $1 \le r \in \mathbb{R}$ is a variable to be optimized and $qn \le x \le n$. This function was designed to possess the following three properties: $C(qn)=0$, $C(n)=1$, and convexity. These

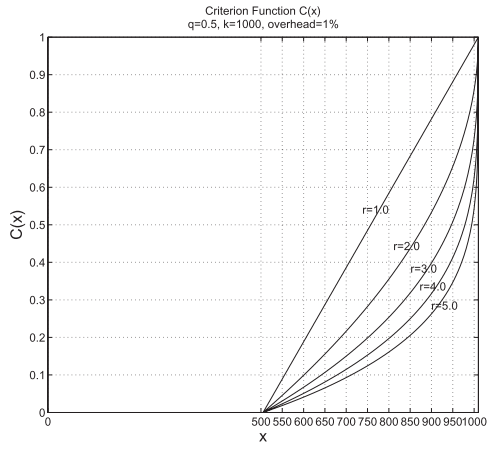

Fig. 1 Plots of $C(x)$ for $q = 0.5$.

properties clearly allow us to assume that if a row of weight two associated with the largest component is chosen for processing whenever $P(x) \leq C(x)$, then we can expect the size of the processed component to be neither large nor small for a suitably chosen r which can be determined through simulations. We will demonstrate the validity of this argument empirically in the next section. By selecting a weight two row for processing in this way, $P(x)$ will look like a staircase function that increases along the curve of $C(x)$. We will call this on-the-fly algorithm method 1. We note that since $P(x)$ and $C(x)$ can both be calculated on-the-fly, the inequality can also be tested on-the-fly. Examples of plots of $C(x)$ for $k = 1000$, $\epsilon = 0.01$ and $q = 0.5$ where $\epsilon \stackrel{\triangle}{=} \frac{n}{k} - 1$ for $r = 1, 2, \cdots, 5$ are shown in Fig. 1.

To prevent the possible case of an abrupt jump in the value of $P(x)$ by method 1 which will occur if the size of a processed component is large, we also tried a second method in which a component is processed if $P(x) \leq C(x)$ but stopped when $P(x) = C(x)$. In other words, in this method, a component chosen for processing may not be completely processed when the associated encoding symbol is received and the remaining set of unprocessed columns, if any, will be processed after the next encoding symbol is received with the same processing criteria. We will call this algorithm method 2. In both methods, all remaining unprocessed columns, if any, are processed after n encoding symbols are received.

4. Results

In this section, we present the elapsed decoding times of method 1 and method 2, and compare them to those using the optimized conventional ML decoding of [2]. All simulations here have been tested on Intel(R) Xeon(R) CPU @1.60 GHz and the results correspond to the case $k = 1000$ and $n = 1010$. Table 1 lists the decoding times elapsed (in milliseconds) of the first and all four phases, and the size of u when $q = 0.5$. The decoding time elapsed of the first phase is defined here as that starting when n encoding symbols have been received for a fair comparison with the conventional decoding. The table shows that as r increases, the size of u decreases, and hence also the combined decoding times of second through fourth phases, as expected. Also shown in the table is the trend that decoding times of the first phase increase as r increases since there remain more unprocessed columns left when n encoding symbols have been received for larger values of r. In total decoding times elapsed, method 1 and method 2 showed the best performance when $r = 3$ and $r = 2$, respectively. We note that these decoding times can be improved by better selection of the value of r which need not be an integer.

Table 1 Decoding times (ms) and size of u when $q = 0.5$.

	$r = 1$	$r = 2$	$r = 3$	$r = 4$	$r = 5$
method 1 (first phase)	0.01	0.01	0.05	0.33	0.56
method 1 (total)	1.21	0.71	**0.61**	0.8	1
method 1 (size of u)	145.99	100.36	82.26	73.44	69.07
method 2 (first phase)	0.04	0.15	0.3	0.45	0.6
method 2 (total)	1.25	**0.85**	0.88	0.95	1.05
method 2 (size of u)	146.17	100.48	82.28	73.53	69.08

Table 2 Decoding times (ms) and size of u when $q = 0.9$.

	$r = 1$	$r = 1.5$	$r = 2$	$r = 2.5$	$r = 3$
method 1 (first phase)	0.01	0.01	0.02	0.12	0.33
method 1 (total)	0.54	0.5	**0.47**	0.55	0.74
method 1 (size of u)	79.77	72.02	68.01	65.65	64.05
method 2 (first phase)	0.07	0.15	0.25	0.35	0.45
method 2 (total)	**0.61**	0.62	0.68	0.78	0.85
method 2 (size of u)	79.8	72.05	68.04	65.66	64.05

Table 3 Decoding times (ms) and size of u of [2].

first phase	1.35	total	1.8	size of u	50.43

Table 2 lists a similar trend of results for $q = 0.9$ case. Here, method 1 and method 2 showed the best total decoding times elapsed when $r = 2$ and $r = 1$, respectively. Tables 1 and 2 indicate that it is not necessarily beneficial to start decoding early as results for $q = 0.9$ case are clearly better than those for $q = 0.5$. This can be explained by the fact that small values of q translate to large sizes of u (a bad effect) but also fast decoding times of the first phase (a good effect) given everything else, e.g., r, held constant. Thus a balance between the two effects must be considered for better overall performance.

Table 3 lists the corresponding results of the optimized conventional ML decoding [2]. Comparing with method 1 ($r = 2$) and method 2 ($r = 1$) of Table 2, the proposed on-the-fly decoding clearly provides reduction in total decoding time by more than $\frac{2}{3}$ over the conventional decoding. The size of u of the on-the-fly decoding algorithm is inherently larger than that of the conventional ML decoding as can be verified in Tables 2 and 3. While this fact is disadvantageous to our proposed on-the-fly decoding algorithm for very large code lengths, for code lengths as those supported in the 3GPP standard [5], the difference in the sizes will be too small to have significant effect in total decoding time elapsed.

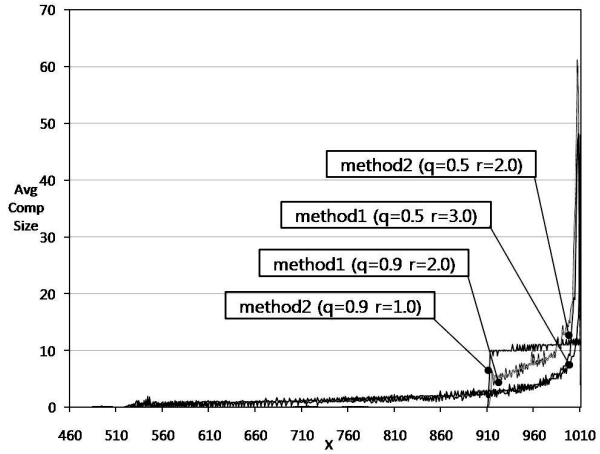

Fig. 2 Average component size.

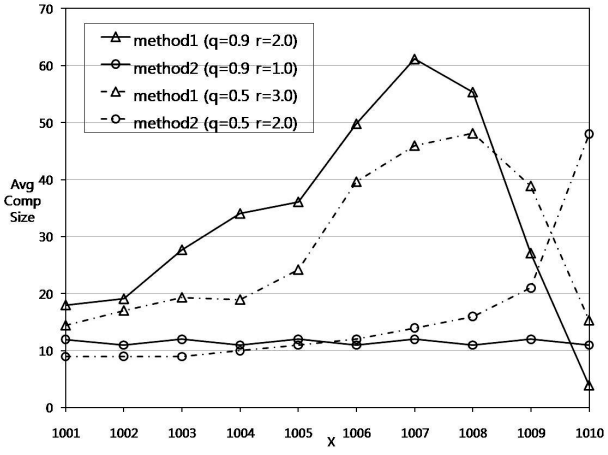

Fig. 3 Average component size.

Figure 2 shows the average sizes of the components processed by the two best methods from each of Tables 1 and 2, and Fig. 3 shows a closer view of the same figure for x close to n. As claimed in the previous section, the size of the processed components is relatively small until x becomes very close to n even after which it does not get too large. In particular, it remains relatively constant and small in method 2 for x less than n as expected. Note that this characteristic of processed components is important since it enables our on-the-fly decoding to be processed concurrently with data reception without delay.

5. Conclusion

We have proposed an efficient on-the-fly algorithm for ML decoding of Raptor codes that can reduce the actual decoding time elapsed by more than $\frac{2}{3}$ with respect to an optimized conventional ML decoding. By properly adjusting the values of q and r in the criterion function, we believe the improvement can be higher than $\frac{2}{3}$.

Acknowledgments

The work of S. Kim was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2010-0016802) and by the Special Research Grant of Sogang University 200911057.01. The work of J. Heo was supported by the IT R&D program of MKE/KEIT (KI001855, Wireless Local Area Communication Systems on Tera Hertz Band).

References

[1] V. Bioglio, M. Grangetto, R. Gaeta, and M. Sereno, "On the fly Gaussian elimination for LT codes," IEEE Commun. Lett., vol.13, no.12, pp.953–955, Dec. 2009.
[2] S. Kim, S. Lee, and S.-Y. Chung, "An efficient algorithm for ML decoding of raptor codes over the binary erasure channel," IEEE Commun. Lett., vol.12, no.8, pp.578–580, Aug. 2008.
[3] M. Luby, "LT codes," Proc. ACM Symposium on Foundations of Computer Science, 2002.
[4] M.A. Shokrollahi, "Raptor codes," IEEE Trans. Inf. Theory, vol.52, no.6, pp.2551–2567, June 2006.
[5] 3GPP TS 26.346 v.7.4.0, Technical Specification Group Services and System Aspects; Multimedia Broadcast/Multicast Services (MBMS); Protocols and Codecs, June 2007.