Modification of the MAP Algorithm for Memory Savings

Inkyu Lee, *Senior Member, IEEE*

***Abstract*—It is usually assumed that all state metric values are necessary in the maximum *a posteriori* (MAP) algorithm in order to compute the *a posteriori* probability (APP) values. This paper extends the mathematical derivation of the original MAP algorithm and shows that the log likelihood values can be computed using only partial state metric values. By processing N stages in a trellis concurrently, the proposed algorithm results in savings in the required memory size and leads to a power efficient implementation of the MAP algorithm in channel decoding. The computational complexity analysis for the proposed algorithm is presented. Especially for the $N = 2$ case, we show that the proposed algorithm halves the memory requirement without increasing the computational complexity.**

***Index Terms*—MAP algorithm, memory savings.**

I. Introduction

THE maximum *a posteriori* (MAP) computing algorithm proposed by Bahl *et al.* [1] is a core element in computing the log likelihood values for many concatenated coding structures that employ an iterative decoding method [2]. It is usually assumed that in order to compute the log likelihood values, all state metric values are necessary in the MAP algorithm [1]. Thus, the state metric values are normally stored in memory. This memory requirement often becomes prohibitive in many practical applications, as excessive memory results in high power consumption for memory access and large memory size.

To alleviate the memory problem in the MAP decoder, several architectural approaches were proposed employing "sliding block" techniques that exploit the self-recovery nature of the Viterbi algorithm [3]–[5]. Another possible approach is recalculation of the unstored state metrics to achieve memory savings at the expense of multiple computations of the state metric values. In contrast to these architectural approaches, the memory redundancy of the MAP algorithm was used to reduce its state memory requirements for nonrecursive convolutional codes [6]. However, this approach is not applicable to recursive shift register processes and, hence, to some advanced coding schemes such as turbo codes [2].

In this paper, we present a modification to the MAP algorithm that extends the mathematical derivation of the original MAP algorithm. We show that the log likelihood values can be computed using only partial state metric values. The proposed algorithm works for nonrecursive as well as recursive codes. As a result, we can reduce both the memory size and the associated power consumption by applying the modified MAP algorithm in computing the reliability values. It will be shown later that, especially when we process two stages in a trellis at the same time, we can reduce the power consumption and the memory size by half without increasing the computational complexity.

The following section describes the MAP algorithm and extends the derivation of computing the log likelihood values. In Section III, we analyze the computational complexity of the proposed scheme. The final section concludes this paper.

Manuscript received June 28, 2003; revised April 5, 2004. This work was supported in part by the Korea Science and Engineering Foundation under Grant R08-2003-000-10761-0 and in part by the University IT Research Center Project. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Dennis R. Morgan.

The author is with the Department of Radio Communications Engineering, Korea University, Seoul, Korea (e-mail: inkyu@korea.ac.kr).

Digital Object Identifier 10.1109/TSP.2004.842195

II. Computation of the Log Likelihood Values

Let us define S_k and y_k as the state of the Markov process at time k and the noisy channel output at time k, respectively. For simplicity, we denote a sequence $\mathbf{y}_m^n$ as $\mathbf{y}_m^n = (y_m, y_{m+1}, \cdots y_n)$.

Assuming a data block of length K, we define the probability functions

$$\begin{aligned}\alpha_s^k &= p\left(S_k = s; \mathbf{y}_1^k\right)\\ \beta_s^k &= p\left(\mathbf{y}_{k+1}^K | S_k = s\right)\\ \gamma_{s',s}^k &= p(S_k = s; y_k | S_{k-1} = s').\end{aligned}$$

The Bahl, Cocke, Jelinek, and Raviv (BCJR) algorithm [1] computes the *a posteriori* probability (APP) $p(S_{k-1} = s'; S_k = s|\mathbf{y}_1^K)$ using the above probabilities. The state metrics α_s^k and β_s^k are computed by the forward and backward recursions as

$$\alpha_s^k = \sum_{s' \in S} \alpha_{s'}^{k-1} \gamma_{s',s}^k$$

where $s' \in S$ is the set of states $S_{k-1} = s'$ that have transitions to the state $S_k = s$, and

$$\beta_{s'}^{k-1} = \sum_{s \in S} \beta_s^k \gamma_{s',s}^k \tag{1}$$

where $s \in S$ is the set of states $S_k = s$ that have transitions from the state $S_{k-1} = s'$.

With the APP value, we can compute the log likelihood ratio (LLR) value for input u_k, which is defined as

$$L(u_k) = \log \frac{p\left(u_k = +1|\mathbf{y}_1^K\right)}{p\left(u_k = -1|\mathbf{y}_1^K\right)} = \log \frac{\sum_{S_+^k} \alpha_{s'}^{k-1} \gamma_{s',s}^k \beta_s^k}{\sum_{S_-^k} \alpha_{s'}^{k-1} \gamma_{s',s}^k \beta_s^k} \tag{2}$$

1053-587X/$20.00 © 2005 IEEE

where S_+^k and S_-^k are the set of state pairs (s', s) corresponding to state transitions from $S_{k-1} = s'$ to $S_k = s$ caused by the input $u_k = +1$ and -1, respectively.

We now extend the original BCJR algorithm one step further by computing $\sigma^k(s'', s', s) = p(S_{k-2} = s''; S_{k-1} = s'; S_k = s; \mathbf{y}_1^K)$. Using the chain rule, it can be shown that

$$\begin{aligned}\sigma^k(s'', s', s) &= p\left(S_{k-2}=s''; \mathbf{y}_1^{k-2}\right) \\ &\quad \cdot \frac{p\left(S_{k-2}=s''; S_{k-1}=s'; \mathbf{y}_1^{k-1}\right)}{p\left(S_{k-2}=s''; \mathbf{y}_1^{k-2}\right)} \\ &\quad \cdot \frac{p\left(S_{k-2}=s''; S_{k-1}=s'; S_k=s; \mathbf{y}_1^{k}\right)}{p\left(S_{k-2}=s''; S_{k-1}=s'; \mathbf{y}_1^{k-1}\right)} \\ &\quad \cdot \frac{p\left(S_{k-2}=s''; S_{k-1}=s'; S_k=s; \mathbf{y}_1^{K}\right)}{p\left(S_{k-2}=s''; S_{k-1}=s'; S_k=s; \mathbf{y}_1^{k}\right)} \\ &= p\left(S_{k-2}=s''; \mathbf{y}_1^{k-2}\right) \\ &\quad \cdot p\left(S_{k-1}=s'; y_{k-1} | S_{k-2}=s''; \mathbf{y}_1^{k-2}\right) \\ &\quad \cdot p\left(S_k=s; y_k | S_{k-2}=s''; S_{k-1}=s'; \mathbf{y}_1^{k-1}\right) \\ &\quad \cdot p\left(\mathbf{y}_{k+1}^K | S_{k-2}=s''; S_{k-1}=s'; S_k=s; \mathbf{y}_1^{k}\right).\end{aligned}$$

Using the Markov property that if S_k is known, events after time k do not depend on $\mathbf{y}_1^k$, the above equation is equal to

$$\begin{aligned}&p\big(S_{k-2}=s''; \mathbf{y}_1^{k-2}\big) p(S_{k-1}=s'; y_{k-1} | S_{k-2}=s'') \\ &\times p(S_k=s; y_k | S_{k-1}=s') p\big(\mathbf{y}_{k+1}^K | S_k=s\big) = \alpha_{s''}^{k-2} \gamma_{s'',s'}^{k-1} \gamma_{s',s}^k \beta_s^k.\end{aligned}$$

We also have

$$\begin{aligned}p\left(u_{k-1}=+1; \mathbf{y}_1^K\right) &= \sum_{(s'',s',s)\in S_+^{k-1}} p\left(S_{k-2}; S_{k-1}; S_k; \mathbf{y}_1^K\right) \\ p\left(u_{k-1}=-1; \mathbf{y}_1^K\right) &= \sum_{(s'',s',s)\in S_-^{k-1}} p\left(S_{k-2}; S_{k-1}; S_k; \mathbf{y}_1^K\right)\end{aligned}$$

where $(s'', s', s) \in S_+^{k-1}$ is the set of state sequences that contain transition segments from $S_{k-2} = s''$ to $S_{k-1} = s'$ caused by the input $u_{k-1} = +1$. S_-^{k-1} is similarly defined for $u_{k-1} = -1$.

The log likelihood values for the input u_{k-1} and u_k can be computed as

$$\begin{aligned}L(u_{k-1}) &= \log \frac{\sum_{S_+^{k-1}} \alpha_{s''}^{k-2} \gamma_{s'',s'}^{k-1} \gamma_{s',s}^k \beta_s^k}{\sum_{S_-^{k-1}} \alpha_{s''}^{k-2} \gamma_{s'',s'}^{k-1} \gamma_{s',s}^k \beta_s^k} \\ L(u_k) &= \log \frac{\sum_{S_+^{k}} \alpha_{s''}^{k-2} \gamma_{s'',s'}^{k-1} \gamma_{s',s}^k \beta_s^k}{\sum_{S_-^{k}} \alpha_{s''}^{k-2} \gamma_{s'',s'}^{k-1} \gamma_{s',s}^k \beta_s^k}.\end{aligned}$$

Note that in the original MAP algorithm, both $\alpha_{s''}^{k-2}$ and $\alpha_{s'}^{k-1}$ are required to compute $L(u_{k-1})$ and $L(u_k)$. However, the modified algorithm can compute $L(u_{k-1})$ and $L(u_k)$ with no knowledge of $\alpha_{s'}^{k-1}$.

By extending the above equations N times, we can generalize the results as

$$\begin{aligned}p\Big(S_{k-N} = s^{(N)}; S_{k-N+1} = s^{(N-1)}; \cdots \\ S_{k-1} = s'; S_k = s; \mathbf{y}_1^K\Big) = \alpha_{s^{(N)}}^{k-N} \Gamma_N^k \beta_s^k\end{aligned}$$

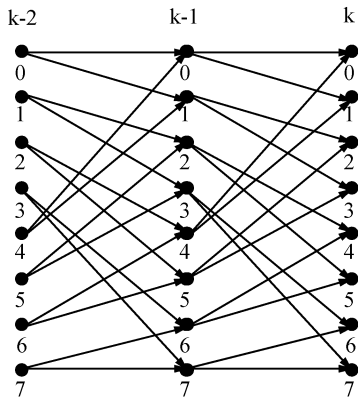

Fig. 1. Eight-state trellis from time $k-2$ to time k.

where the total branch metric Γ_N^k is defined as $\Gamma_N^k = \gamma_{s^{(N)},s^{(N-1)}}^{k-N+1} \gamma_{s^{(N-1)},s^{(N-2)}}^{k-N+2} \cdots \gamma_{s^{(1)},s^{(0)}}^k$.

As an example, Fig. 1 shows an eight-state trellis from time $k-2$ to time k for $N = 2$. The number next to the state specifies the state number. For example, the total branch metric Γ_2^k from state 2 at time $k-2$ to state 0 at time k is equal to $\gamma_{2,4}^{k-1} \gamma_{4,0}^k$. Note that for a 2^M-state trellis, the path between time $k-N$ to time k is uniquely defined for $N \le M$ as M determines the number of memory elements in the shift register. For $N > M$, there exist multiple paths between two states.

Thus, it follows that the log likelihood values for any N value are obtained as

$$L(u_{k-i}) = \log \frac{\sum_{S_+^{k-i}} \alpha_{s^{(N)}}^{k-N} \Gamma_N^k \beta_s^k}{\sum_{S_-^{k-i}} \alpha_{s^{(N)}}^{k-N} \Gamma_N^k \beta_s^k}, \text{ for } i = 0, \mathbf{1}, \cdots N-1 \tag{3}$$

where S_+^{k-i} and S_-^{k-i} are the set of state sequences that contain transition segments caused by the input $u_{k-i} = +1$ and $u_{k-i} = -1$, respectively.

This clearly shows that the intermediate state metric values $\alpha_{s'}^{k-1}, \alpha_{s''}^{k-2}, \cdots \alpha_{s^{(N-1)}}^{k-N+1}$ are not necessary for computing $L(u_{k-i})$ for $i = 0, 1, \cdots N-1$.[1] Thus, the LLR values $L(u_{k-i})$ can be computed using only $\alpha_{s^{(N)}}^{k-N}$ and β_s^k. This leads to savings of the required state metric memory size by N times. Note that the branch metric memory size is not reduced. To the best of the author's knowledge, the derivation of (3) has yet to be addressed in the literature. Hoeher [7] proposed a subblock-based scheme, applying a similar block processing technique. However, the individual LLR values were not computed.

Moreover, the proposed algorithm allows us to reduce the power consumption N times with respect to accessing the state metric memory since data access is carried out at frequency N times lower than the conventional MAP algorithm. It was analyzed [8], [9] that data storage and transfer operations normally consume much more power than a data computation operation. Note that in the sliding window architectures [3]–[5], it is not possible to achieve such a power reduction since the memory access rate is unchanged. In the following section, the complexity issues associated with the proposed algorithm will be discussed.

[1]Depending on the implementation architecture, β_s^k's can be chosen to be stored instead of α_s^k's.

TABLE I
NUMBER OF MULTIPLICATIONS AND ADDITIONS FOR THE PROPOSED ALGORITHM

multiplications	$2^{M+N}+N$
additions	$(2^M+N-1)2^N-2N$

III. COMPUTATIONAL COMPLEXITY

We now consider the complexity issue in this section. Worm *et al.* recognized a high power consumption problem in a turbo decoder design associated with the memory accessing scheme [8]. In a straightforward fashion, the APP computation in (2) requires $2 \cdot 2 \cdot 2^M$ multiplications. Note that the computations for the forward and backward recursions are not included here. However, it was noted [10] that the $\beta_s^k \gamma_{s',s}^k$ term in (1) is also used in the APP computation in (2). Thus, while the $\beta_{s'}^{k-1}$'s are being calculated, the APP values should be computed at the same time, reusing the $\beta_s^k \gamma_{s',s}^k$ terms to minimize the number of computations. Therefore, the total number of multiplications reduces to $2 \cdot 2^M$.

By extending this idea to the proposed algorithm, we can further reduce the computational complexity in the modified MAP algorithm. We note that the $\Gamma_N^k \beta_s^k$ term in (3) is already available when the state metric $\beta_{s(N)}^{k-N}$'s are computed. Thus, the number of multiplications in (3) becomes $2^M \cdot 2^N$ when N is not greater than M. In comparison, the conventional MAP algorithm requires $2^M \cdot 2N$ for an N-stage trellis. Note that regardless of N, the total number of multiplications required for computing $\beta_{s(N)}^{k-N}$ remains the same. Thus, the exact formulas for the number of multiplications including N divisions is $2^{M+N}+N$.

As for additions, the additions are first grouped into 2^N sets, each set adding 2^M paths corresponding to the N bit data sequence through the trellis, for a total of $(2^M-1) \cdot 2^N$ additions. Then, 2^{N-1} sets are added together $2N$ times, corresponding to one of the N bits being determined. This requires $(2^{N-1}-1) \cdot 2N$ additions. Thus, the total number of additions is given by $(2^M+N-1) \cdot 2^N - 2N$. This is summarized in Table I, which also includes the number of multiplications $2^{M+N}+N$ and additions $2N(2^M-1)$ required for the standard MAP algorithm for comparison.

As an example, in the eight-state trellis case $(M=3)$ shown in Fig. 1, the original MAP algorithm requires $2 \cdot 17 = 34$ multiplications and $2 \cdot 14 = 28$ additions in computing $L(u_{k-1})$ and $L(u_k)$. In comparison, the new derivation for the $N=2$ case also requires 34 multiplications and 32 additions, which is almost the same as the original MAP algorithm. When we extend the trellis one stage further $(N=3)$, the original algorithm needs $3 \cdot 17 = 51$ multiplications and $3 \cdot 14 = 42$ additions, whereas the proposed algorithm requires 67 multiplications and 74 additions, which is a 31% increase in multiplications and a 76% increase in additions compared with the original algorithm. Even in this case, considering that computing $L(u_k)$ takes about half of the total computation of the whole MAP algorithm, including the forward and backward recursions [9], the overall increased computations for multiplications and additions are roughly 16% and 38%, respectively, and in return, we reduce the memory space and associated power consumption to one third of their original values. Please note that the multiplication operation can be easily transformed into the addition operation by applying the log MAP algorithm.

TABLE II
COMPARISON OF MEMORY SAVINGS AND COMPUTATIONAL COMPLEXITY

N	memory savings	increase in computational complexity
1	1	1
2	1/2	1
3	1/3	4/3
4	1/4	2

It should be mentioned that even though the memory savings become greater with larger N, N cannot be increased arbitrarily. As an extreme case, when N is set to the maximum possible value K—the length of the entire data block—no intermediate state metric values need to be stored. However, the associated computational complexity would be prohibitive. Therefore, a designer should choose N, which optimizes both the memory savings $(1/N)$ and the computational complexity increase $(2^{N-1}/N)$. The tradeoff between these two aspects was studied in [11].

Table II summarizes rough estimates for the memory savings and the computational complexity, assuming large M values. The $N=1$ case refers to the original MAP algorithm. This table clearly demonstrates that the proposed algorithm becomes an attractive approach for small N in terms of power and memory efficient implementation. Please note that the actual tradeoff between the memory savings and the computational complexity would depend on the technology used for the chip implementation.

IV. CONCLUSION

In this paper, we extend the original MAP algorithm and show that the APP values can be computed by a part of all state metric values. The proposed algorithm reduces the required state metric memory size by a factor of N. Especially for the $N=2$ case, we can halve the power consumption related to memory access and the memory size without incurring an increase in computational complexity. As the power consumption associated with memory access is reduced significantly, the proposed algorithm can be employed by applications such as handset terminals, where minimizing the total power consumption is crucial. In addition, it should be mentioned that we can combine the proposed algorithm with other memory-saving techniques in turbo coding applications to further reduce the memory size.

ACKNOWLEDGMENT

The author would like to thank the anonymous reviewer for the equations in Table I.

REFERENCES

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

[2] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: turbo-codes," in *Proc. IEEE Int. Conf. Commun.*, May 1993, pp. 1064–1070.

[3] A. J. Viterbi, "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes," *IEEE J. Sel. Areas Commun.*, vol. 16, no. 2, pp. 260–264, Feb. 1998.

[4] S. S. Pietrobon, "Efficient implementation of continuous MAP decoders and a synchronization technique for turbo decoders," in *Proc. Int. Symp. Inform. Theory Appl.*, 1996, pp. 586–589.

[5] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Soft-output decoding algorithms for continuous decoding of parallel concatenated convolutional codes," in *Proc. IEEE Int. Conf. Commun.*, June 1996, pp. 112–117.

[6] M. Schmidt and G. P. Fettweis, "On memory redundancy in the BCJR algorithm for nonrecursive shift register processes," *IEEE Trans. Inf. Theory*, vol. 46, no. 4, pp. 1580–1584, Jul. 2000.

[7] P. Hoeher, "Optimal subblock-by-subblock detection," *IEEE Trans. Commun.*, vol. 43, no. 2/3/4, pp. 714–717, Feb/Mar/Apr 1995.

[8] A. Worm, H. Michel, and N. When, "Power minimization by optimizing data-transfers in turbo-decoders," *Kleinheubacher Berichte*, pp. 343–350, Sep. 1999.

[9] M. L. Vallejo, S. A. Mujtaba, and I. Lee, "A low-power architecture for maximum a posteriori turbo-decoding," in *Proc. 36th Asilomar Conf.*, Nov. 2002, pp. 47–51.

[10] S. S. Pietrobon, "Implementation and performance of a turbo/MAP decoder," *Int. j. Satellite Commun.*, vol. 16, pp. 23–46, Jan.–Feb. 1998.

[11] I. Lee, M. L. Vallejo, and S. A. Mujtaba, "Block processing technique for low power turbo decoder design," in *Proc. Veh. Technol. Conf.*, Apr. 2002, pp. 1025–1029.

Inkyu Lee (S'91–M'95–SM'01) received the B.S. degree (with honors) in control and instrumentation engineering from Seoul National University, Seoul, Korea, in 1990 and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1992 and 1995, respectively.

From 1991 to 1995, he was a Research Assistant at the Information Systems Laboratory, Stanford University. From 1995 to 2001, he was a Member of Technical Staff, Bell Laboratories, Lucent Technologies, Murray Hill, NJ, where he studied wireless communication system design. He later worked for Agere Systems (formerly the Microelectronics Group of Lucent Technologies) from 2001 to 2002. In September 2002, he joined the Faculty of the Department of Radio Communications Engineering, Korea University, Seoul.

Dr. Lee received the Young Scientist Award from the Ministry of Science and Technology of Korea in 2003. He has been an Editor of wireless communications theory for the IEEE TRANSACTIONS ON COMMUNICATIONS since May 2001. He is also Editor-in-Chief of the Special Issue on 4G Wireless Systems of the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS.