An Intuitive Justification and a Simplified Implementation of the MAP Decoder for Convolutional Codes

Andrew J. Viterbi, *Life Fellow, IEEE*

***Abstract*— An intuitive shortcut to understanding the maximum *a posteriori* (MAP) decoder is presented based on an approximation. This is shown to correspond to a dual-maxima computation combined with forward and backward recursions of Viterbi algorithm computations. The logarithmic version of the MAP algorithm can similarly be reduced to the same form by applying the same approximation. Conversely, if a correction term is added to the approximation, the exact MAP algorithm is recovered. It is also shown how the MAP decoder memory can be drastically reduced at the cost of a modest increase in processing speed.**

***Index Terms*—Dual-maxima, MAP decoder, soft-decision metric, Viterbi algorithm.**

I. Introduction

THE maximum *a posteriori* (MAP) decoding algorithm for convolutional codes was proposed over two decades ago[1] by Bahl *et al.* [1], but initially received very little attention because of its increased complexity over alternative convolutional decoders for a minimal advantage in bit-error rate performance. Recently, however, the MAP decoder has enjoyed renewed and greatly increased attention as an iterative soft-output decoder for the class of "turbo" codes discovered by Berrou *et al.* [3], as well as the class of serial concatenated codes with random interleaving, more recently proposed, analyzed, and simulated by Benedetto *et al.* [4]. Other soft-output decoding algorithms have also been proposed and successfully demonstrated, notably the SOVA algorithm of Hagenauer and Hoeher [5]. Only the MAP algorithm, however, achieves acceptable performance at E_b/N_0 levels within 1 dB of the value which corresponds to Shannon capacity.

In the next section, we derive intuitively an approximation to the MAP algorithm. We then proceed to justify this heuristic approach by demonstrating its structural equivalence to the formally derived MAP algorithm, particularly after the same approximation is applied to the basic function required to implement the latter. The connection is further strengthened by showing that when the approximation is augmented by a correction term, it becomes the same as the logarithmic form of the MAP algorithm. In the last section, we consider implementation complexity, and show that the MAP decoder can be implemented with no more than four times the complexity of a Viterbi decoder for the same code.

Manuscript received July 19, 1996; revised August 5, 1997. This paper was presented at the IEEE Information Theory Workshop, Haifa, Israel, June 1996.

The author is with QUALCOMM Inc., San Diego, CA 92121 USA.

Publisher Item Identifier S 0733-8716(98)00165-6.

[1] A similar forward–backward recursion algorithm for MAP demodulation for channels with intersymbol interference had previously been published by Chang and Hancock [2].

II. An Intuitive Approximate Approach to the MAP Algorithm

Let us first consider briefly the MAP[2] soft-output decoder for a block code. If $\boldsymbol{u}$ is the information bit sequence and $\boldsymbol{y}$ is the sequence of received channel output symbols, then the *a posteriori* probability ratio for the kth bit is just

$$\Lambda_k = \frac{P(u_k=+1|\boldsymbol{y})}{P(u_k=-1|\boldsymbol{y})} = \frac{\sum_{\boldsymbol{u}:\,u_k=+1} p(\boldsymbol{u},\boldsymbol{y})}{\sum_{\boldsymbol{u}:\,u_k=-1} p(\boldsymbol{u},\boldsymbol{y})}. \tag{1}$$

In preference to using Λ_k as the soft-decision output, its logarithm has the advantage that, for a memoryless channel, the overall metric can be formed as sums, rather than products, of independent components or metrics. Thus defining

$$L_k \triangleq \ln \Lambda_k \tag{2}$$

for the soft-output metric, and

$$M(\boldsymbol{u},\boldsymbol{y}) \triangleq \ln p(\boldsymbol{u},\boldsymbol{y}) = \ln P(\boldsymbol{u}) + \ln p(\boldsymbol{y}|\boldsymbol{u}) \tag{3}$$

it follows from (1)–(3) that

$$L_k = \ln \sum_{\boldsymbol{u}:\,u_k=+1} e^{M(\boldsymbol{u},\boldsymbol{y})} - \ln \sum_{\boldsymbol{u}:\,u_k=-1} e^{M(\boldsymbol{u},\boldsymbol{y})}. \tag{4}$$

Exponentiation enhances the differences between individual metrics $M(\boldsymbol{u},\boldsymbol{y})$. Hence, typically, one term will dominate each sum, which suggests the approximation

$$\ln \sum_j e^{a_j} \approx \max_j a_j \tag{5}$$

which is obviously also a lower bound. When applied to (4), this yields

$$L_k \approx \max_{\boldsymbol{u}:\,u_k=+1} M(\boldsymbol{u},\boldsymbol{y}) - \max_{\boldsymbol{u}:\,u_k=-1} M(\boldsymbol{u},\boldsymbol{y}) \tag{6}$$

a metric which has been termed the "dual-maxima" rule for block codes [6], [7].

[2] In the context of soft-output decoding which goes beyond the original goal of Bahl *et al.* [1] of merely minimizing the bit-error probability, a more appropriate term for this decoder would be just the *a posteriori* probability (APP) computation algorithm.

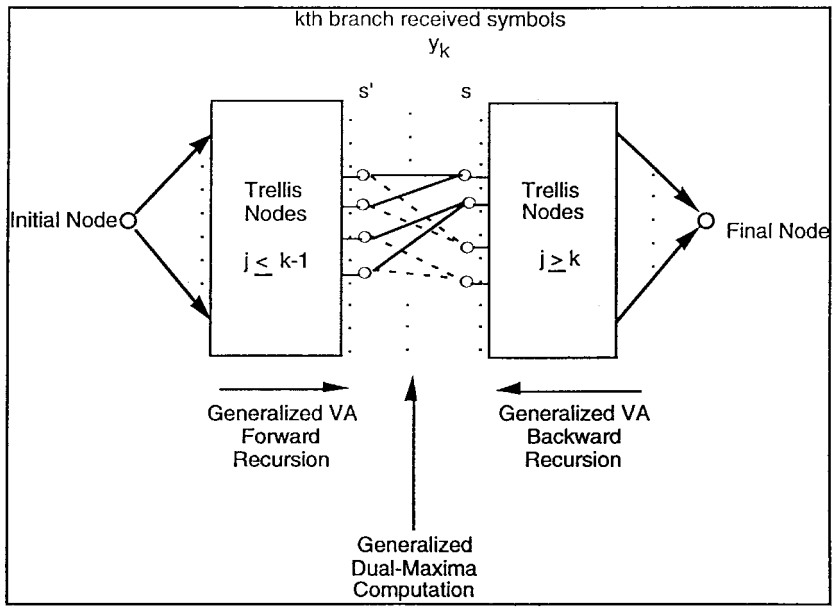

Fig. 1. MAP framework.

We now proceed to apply the same approach to a block code generated as a convolutional code which is truncated by forcing the encoder to a known (e.g., $\mathbf{0}$) state as shown in Fig. 1. While (4) and (6) still apply, they involve an inordinately complex computation involving 2^N terms, where N is the length of the information bit sequence. We proceed to simplify (6) by recognizing that the maximum metric for a given state of the convolutional code's trellis can be obtained from the conventional Viterbi algorithm (VA). Suppose then that we seek L_k for the kth bit, when the *channel is memoryless*. For simplicity, let the code trellis be binary, so that one branch corresponds to a single bit (this can be easily generalized to treat multiple bits per branch). Then we may argue intuitively as follows, referring to Fig. 1. Let us generate all of the state metrics at the $(k-1)$th node by applying the Viterbi algorithm (VA) from the initial node to this point. For each state, the metric corresponds to the maximum metric over all paths up to that node. Let us denote the state metrics at the $(k-1)$th node $a_{k-1}(s')$ and those at the kth node $a_k(s)$, where s' and s are the generic states at the $(k-1)$th and kth nodes, respectively. These are generated by the Viterbi algorithm, whose definition is the recursion relation

$$a_k(s) = \max_{s'}[a_{k-1}(s') + c_k(s', s)]; \quad a_0(\mathbf{0}) = 0;$$
$$a_0(s) = -\infty, \quad s \neq \mathbf{0} \tag{7}$$

where $c_k(s', s)$ is the branch metric for the branch connecting state s' at node $k-1$ to state s at node k. The state metrics for the portion of the trellis beyond the kth node can similarly be computed recursively by a backward VA starting at the last node (Fig. 1). Thus denoting the generic states at nodes $j-1$ and j, s', and s, respectively, and the corresponding state metrics $b_{j-1}(s')$ and $b_j(s)$, the recursion for the backward VA can be similarly stated as

$$b_{j-1}(s') = \max_{s}[b_j(s) + c_j(s', s)]; \quad b_N(\mathbf{0}) = 0;$$
$$b_N(s) = -\infty, \quad s \neq \mathbf{0} \tag{8}$$

where $c_j(s', s)$ is again the metric from the branch connecting s' to s. This branch metric required by both recursions is just the log-likelihood function

$$c_k(s', s) \triangleq \ln p(y_k|s, s') + \ln P(s|s'). \tag{9}$$

Thus, for two states s' and s, for which a branch transition does not exist, the metric is negative infinity, as seen from (9). However, for all pairs (s', s) for which a branch transition exists, we may combine the forward state metrics at node $k-1$, the backward state metrics at node k, and the metrics for branches connecting the two sets to obtain for the approximate metric of (6) the expression[3]

$$L_k \approx \max_{s', s: u_k=+1}[a_{k-1}(s') + c_k(s', s) + b_k(s)] - \max_{s', s: u_k=-1}[a_{k-1}(s') + c_k(s', s) + b_k(s)]. \tag{10}$$

Note that the first maximum is over all branch pairs, s' at node $k-1$ and s at node k for which the connecting branch is shown as a solid line in Fig. 1, while the second maximum is over all pairs for which the connecting branch is shown as a dotted line.

[3]It was pointed out by a reviewer that a similar structure was previously proposed in [8]. However, the reference did not show the evolution of the approximate from the exact formulation nor the detailed relationship between the two, as described in this and the next sections.

We next show that this result can be reached by applying the approximation (5) to the exact formulation of the MAP decoder; conversely, we shall also show that the exact result can be obtained by applying a correction term to the maximum functions of (7), (8), and (10).

III. Relationships of the Approximate to the Exact Formulation

Hagenauer *et al.* [9] obtained an elegant derivation of the MAP decoder by partitioning the joint probability

$$\begin{aligned} p(s', s, \boldsymbol{y}) &= p(s', s, \boldsymbol{y}_{j<k}, \boldsymbol{y}_k, \boldsymbol{y}_{j>k}) \\ &= p(s', \boldsymbol{y}_{j<k}) p(\boldsymbol{y}_k, s | s') p(\boldsymbol{y}_{j>k} | s) \triangleq \boldsymbol{e}^{\hat{a}_k} \boldsymbol{e}^{c_k} \boldsymbol{e}^{\hat{b}_k} \end{aligned}$$

where $\boldsymbol{y}_{j<k}$ and $\boldsymbol{y}_{j>k}$ are the sequences of received symbols before and after the kth branch. Replacing the summations in the numerator and denominator of (1) by the summations over all state pairs (s', s) for which u_k is +1 and -1, respectively, one obtains for the logarithm of (1)

$$\begin{aligned} L_k = &\ln \sum_{s', s: u_k = +1} \exp\left(\hat{a}_{k-1} + c_k + \hat{b}_k\right) \\ &- \ln \sum_{s', s: u_k = -1} \exp\left(\hat{a}_{k-1} + c_k + \hat{b}_k\right) \end{aligned} \tag{10'}$$

with the recursions for $\hat{a}_k$ and $\hat{b}_j$

$$\begin{aligned} &\hat{a}_k(s) = \ln \sum_{s'} \exp[\hat{a}_{k-1}(s') + c_k(s', s)]; \\ &\hat{a}_0(\mathbf{0}) = 0; \quad \hat{a}_0(s) = -\infty, \quad s \neq \mathbf{0} \end{aligned} \tag{7'}$$

$$\begin{aligned} &\hat{b}_{j-1}(s') = \ln \sum_{s} \exp[c_j(s', s) + \hat{b}_j(s)]; \\ &b_N(\mathbf{0}) = 0; \quad b_N(s) = -\infty, \quad s \neq \mathbf{0} \end{aligned} \tag{8'}$$

and where c_k is again the branch metric given by (9). Clearly, primed (7′), (8′), and (10′) become the same as their unprimed counterparts, developed intuitively, if we use (5) to approximate the logarithm of the sum-of-exponentials by the maximum.

More interesting, however, is to apply the reverse process to the approximate development of the previous section. Following several authors [10]–[13], we define the function

$$\max^*(x, y) \triangleq \max(x, y) + \ln\left(1 + \boldsymbol{e}^{-|x-y|}\right). \tag{11}$$

It follows from the definition that

$$\max^*(x, y) = \ln(\boldsymbol{e}^x + \boldsymbol{e}^y).$$

It also follows that just as

$$\max(x, y, z) = \max[\max(x, y), z]$$

so also

$$\begin{aligned} \max^*(x, y, z) &= \max^*[\max^*(x, y), z] \\ &= \ln(\boldsymbol{e}^x + \boldsymbol{e}^y + \boldsymbol{e}^z). \end{aligned} \tag{12}$$

Thus, replacing max by max* in the approximate expressions (7), (8), and (10), we obtain the exact expressions (7′), (8′), and (10′). This justifies our labeling in Fig. 1 the forward and backward recursions as "generalized Viterbi algorithms" and the computation of L_k as a "generalized dual-maxima" computation.

We note finally, as has been observed in [11] and [12], that the implementation of $\max^*(x, y)$ is only slightly more complex than the implementation of $\max(x, y)$. The latter requires a subtractor to form $(x - y)$ followed by a comparator with zero, while $\max^*(x, y)$ requires additionally only a read-only memory (ROM) which outputs the correction term $\ln(1 + \boldsymbol{e}^{|-x-y|})$ given the input $x - y$, which is the subtractor output. This also shows that, just as for a Viterbi decoder, a common term can be subtracted from all metrics at a given node to avoid overflows, with no consequence to performance.

IV. Implementation for Memory Reduction

Now that the implementation has been reduced to a series of common decoder operations, the obvious remaining drawback of the MAP algorithm is the excessive memory required. As described in the above, the entire state metric history must be stored, out to the end of trellis, at which point the backward algorithm begins and decisions can be output starting with the last branch, without the need to store any but the last set of state metrics computed backward. This storage requirement is obviously excessive; for a 16-state code, assuming 6-bit state metrics, it would require 96 bits of storage per branch, for a total of 96 000 bits for a 1000-bit block, judged to be minimal for turbo code performance.

We now describe a technique[4] which reduces the memory requirement for a 16-state code to just a few thousand bits, independent of the block length. It can best be described by referring to the timing diagram of Fig. 2, which indicates the bit processing times for one forward processor and two backward processors operating in synchronism with the received branch symbols, i.e., computing one set of state metrics during each received branch time (bit time for a binary trellis).

The basis for this approach is the fact that the VA can start cold in any state at any time; initially, the state metrics generated are nearly worthless, but after a few constraint lengths, the set of state metrics are as reliable as if the process had been started at the initial (or final) node. Let this "learning" period consist of L branches. (For a 16-state code, $L = 32$ is more than sufficient, amounting to over six constraint lengths of the convolutional code.) This applies equally to the backward as well as the forward algorithm, and assumes that all state metrics are normalized by subtracting at every node an equal amount from each.

Let the received branch symbols be delayed by $2L$ branch times. Then the forward algorithm processor starts at the initial node at branch time $2L$, computing all state metrics for each node every branch time and storing these in memory. The first backward processor starts at the same time, but processes backward from the $2L$th node, setting every initial state metric to the same value, not storing anything until branch time $3L$, at which point it has built up reliable state metrics and it

[4] This approach is similar to the "sliding block" techniques of Barbulescu [14], Pietrobon [15] and Benedetto *et al.* [13], but differs in the timing and storage requirements and the coordination of the forward and backward processing.

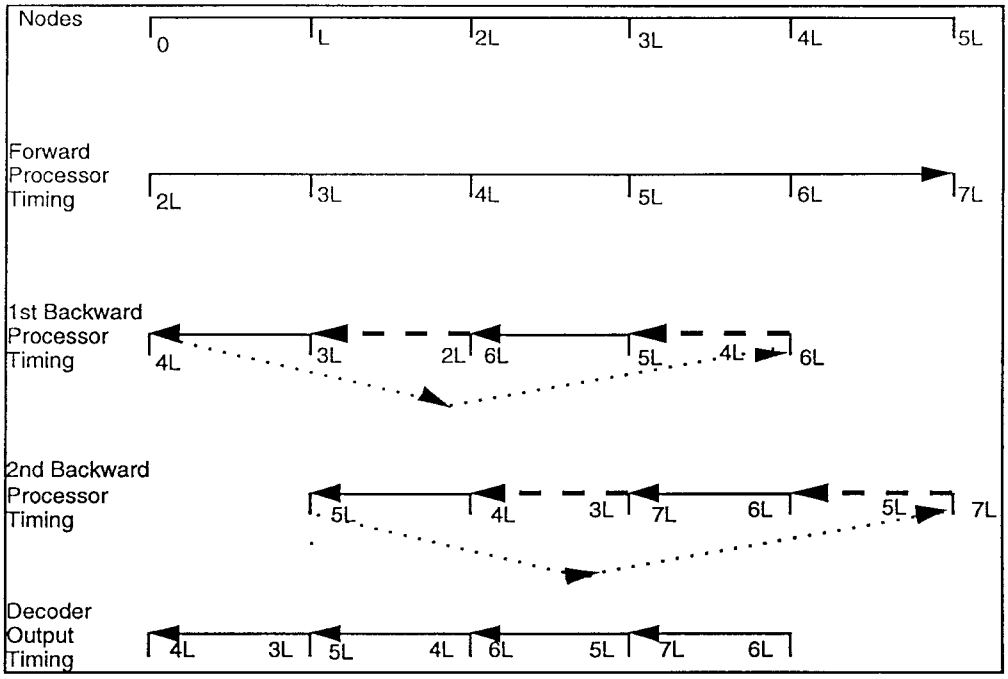

Fig. 2. Processor's timing.

encounters the last of the first set of L forward computed metrics. (In Fig. 2, the top line indicates the node indexes; the remaining lines are labeled according to the times at which the branches are processed. Also, unreliable metric branch computations are shown as dashed lines.) At this point, the generalized dual-maxima process is performed according to (10′), the Lth branch soft decisions are output, and the backward processor proceeds until it reaches the initial node at time $4L$. Meanwhile, starting at time $3L$, the *second* backward processor begins processing with equal metrics at node $3L$, discarding all metrics until time $4L$, when it encounters the forward algorithm having computed the state metrics for the $2L$th node. The generalized dual-maxima process is then turned on until time $5L$, at which point all soft decision outputs from the $2L$th to the Lth node will have been output. The two backward processors hop forward $4L$ branches every time they have generated backward $2L$ sets of state metrics, and they time share the output processor since one generates useless metrics while the other generates the useful metrics which are combined with those of the forward algorithm.

Note that nothing needs to be stored for the backward algorithms except for the metric set of the last node, and these only when reliable metrics are being generated. The forward algorithm only needs to store $2L$ sets of state metrics[5] since, after its first $2L$ computations (performed by time $4L$), its first set of metrics will be discarded, and the emptied storage can then be filled starting with the forward-computed metrics for the $(2L+1)$th node (at branch time $4L+1$). Thus, the storage requirements for a 16-state code using 6-bit state metrics is just $192L$ bits in all, which for $L = 32$ amounts to approximately $6K$ bits. (Note that a conventional $K = 7$ Viterbi decoder with 64 states and a 32-bit path memory requires about $2K$ bits of memory, while a $K = 9$ decoder requires at least a 40-bit path memory resulting in over $10K$ of storage.) We conclude that these storage requirements are no greater than those of a conventional VA for commonly used codes.

As for processing requirements, it would appear that the VA load is thus tripled; furthermore, the complexity of the generalized dual-maxima process is no greater than that of the forward or backward VA processor so that, overall, the complexity is not more than quadrupled—also, the chain-back procedure is avoided. Further, since the code is shorter, the number of states is much reduced relative to the $K = 7$ and 9 examples just given. Since the MAP decoder (with short constraint length) is only justified for iterative decoding of turbo or serially concatenated codes, we must also account for the required number of iterations, which are on the order of 4–8. Thus, a pair of 16-state concatenated decoders performing four iterations imposes double the processing load of a $K = 7$ Viterbi decoder; a pair of four-state concatenated decoders performing eight iterations imposes the same load as a $K = 9$ decoder.

Minimum decoding delay is set by the length of the block or its corresponding interleaver. If the processors described above operate at just the speed of the received branches, it is necessary to pipeline the successive iterations, and hence multiply the minimum delay by the number of iterations. If, on the other hand, the processors can operate at a much higher speed, then additional delay can be much reduced.

V. Concluding Remarks

One purpose of this paper is to clarify and simplify the topic of MAP decoders of convolutional codes, which is often clouded by unintuitive presentations, and hence appears more

[5] Actually, storage for L sets of state metrics would suffice if the storage process were to proceed alternately forward and backward every L branches, but with some added complexity in the process and its description.

complex than it actually is. By its inherent equivalence to a combination of forward and backward VA processors, coupled by a dual-maxima computation, the appearance of complexity is dispelled and quantitatively bounded. Another purpose is to assess implementation complexity. By applying memory management techniques similar to those used for ordinary convolutional decoding, we have bounded the processing load at no more than four times that of a conventional decoder for the same code, with moderate memory requirements. For turbo (parallel) and serially concatenated codes, employing iterative soft-output decoders, the component code constraint lengths are much shorter, which affords the possibility of performing several decoding iterations without exceeding the processing time of a single conventional decoder for the longer constraint lengths in common practice. All of this guarantees the feasibility of such decoders operating at multimegabit per second data rates.

REFERENCES

[1] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284–287, 1974.

[2] R. W. Chang and J. C. Hancock, "On receiver structures for channels having memory," *IEEE Trans. Inform. Theory*, vol. IT-12, pp. 463–468, 1966.

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

[4] S. Benedetto, G. Montorsi, D. Divsalar, and F. Pollara, "Serial concatenation of intereleaved codes: Performance analysis, design and iterative decoding," in *TDA Progr. Rep.* 42-126, Jet Propulsion Lab., Pasadena, CA, pp. 1–26, 1996.

[5] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," in *Proc. IEEE GLOBECOM*, Dallas, TX, 1989, pp. 47.1.1–47.1.7.

[6] A. M. Viterbi and A. J. Viterbi, "Noncoherent receiver employing a dual-maxima metric generation process," U.S. Patent 5 442 627, 1995.

[7] A. J. Viterbi, *CDMA: Principles of Spread Spectrum Communications.* Reading, MA: Addison-Wesley, 1995, ch. 4, pp. 77–121.

[8] Y. Li and B. Vucetic, "A generalized MLSE algorithm," in *Proc. Int. Conf. Neural Networks and Signal Processing, INNSP'95*, China, 1995, pp. 718–721.

[9] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 429–445, 1996.

[10] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in *Proc. ICC'95*, Seattle, WA, 1995, pp. 1009–1013.

[11] S. S. Pietrobon, "Implementation and performance of a serial MAP decoder for use in an iterative turbo decoder," in *Proc. IEEE Int. Symp. Inform. Theory*, Whistler, B.C., Canada, 1995, p. 471.

[12] S. Benedetto, G. Montorsi, D. Divsalar, and F. Pollara, "Soft-output decoding algorithms in iterative decoding of turbo codes," *JPL TDA Progr. Rep.*, vol. 42-124, pp. 63–87, 1995.

[13] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Soft input soft output MAP module to decode parallel and serial concatenated codes," in *TDA Progr. Rep.* 42-127, Jet Propulsion Lab., Pasadena, CA, pp. 1–20, 1996.

[14] S. A. Barbulescu, "Iterative decoding of turbo codes and other concatenated codes," Ph.D. dissertation, Univ. South Australia, pp. 23–24, 1996.

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

Andrew J. Viterbi (S'54–M'58–SM'63–F'73–LF'97) is Vice Chairman and cofounder of QUALCOMM Incorporated. He has spent equal portions of his career in industry, having previously cofounded LINKABIT Corporation, and in academia as Professor in the Schools of Engineering at both University of California at Los Angeles and University of California at San Diego, at which he is now Professor Emeritus.

Dr. Viterbi is a member of both the National Academy of Engineering and the National Academy of Sciences, among the numerous honors he has received.