On Low Complexity Stack Decoding of Convolutional Codes

J. D'Souza and S. L. Maskara

***Abstract*— This letter presents techniques for improving the distribution of the number of stack entries, for stack sequential decoding over hard quantized channel, with emphasis on high rate codes. It is shown that, for a class of high rate $b/(b+1)$ codes, a table-based true high rate approach can be easily implemented for obtaining decoding advantage over the punctured approach. Modified algorithms, which significantly improve the distribution of the number of stack entries and decoding time, are proposed for rate $1/N$ codes and high rate $b/(b+1)$ codes.**

***Index Terms*—Convolutional codes, sequential decoding, stack algorithm.**

I. Introduction

FOR a stack sequential decoder [1], efforts have been made to improve the distribution of the number of stack entries [2]–[4]. This is to reduce the probability of losing the true path node from the stack [2]. Further, since stack reorganization is a time consuming operation, an improvement in decoding speed can also be expected from improvement in the distribution of the number of stack entries [2]. In this letter, we have shown that, for a class of high rate $b/(b+1)$ codes, a table-based true high rate (THR) approach can be easily implemented for hard quantized channel, to obtain decoding advantage over the punctured approach [5], [6]. For this channel, we have also proposed modified algorithms that use tags to give improved distribution of the number of stack entries and decoding time. Our modifications are based on the observation that a node with a smaller metric can get extended only after a node with a better metric reaches the stack top, and consequently it would suffice if the former is introduced into the stack just when the latter reaches the stack top.

II. Table-Based Approach for High Rate Codes

In the context of saving memory, a table-based decoding approach was suggested by Jelinek [2], wherein all the forward branches of the top node are not introduced simultaneously; rather, the next best sibling of the top node is introduced into the stack, and the top node is replaced by its best forward branch (BFB). The complexity of this approach then depends on the ease with which these two operations can be performed. In this section we show that, for a class of rate $b/(b+1)$ THR codes, these two operations are rather simple. Such a table-based decoder will be called Jelinek THR decoder.

A. Best Forward Branch

For the rate $b/(b+1)$ long THR codes listed in the literature, the zeroth-order generator [1] has the form $\mathbf{G}_0 = [\mathbf{S} \quad \mathbf{P}]$, where $\mathbf{P}$ is a $b \times 1$ matrix, and $\mathbf{S}$ is a $b \times b$ matrix for which all the elements below the diagonal are zero and the diagonal elements are all one. This structure of $\mathbf{S}$ implies that, if the message word $\mathbf{m} = (m_1, m_2, \cdots, m_b)$ produces the code word $\mathbf{c} = (c_1, c_2, \cdots, c_{(b+1)})$, then $\mathbf{c}_i$ does not depend on m_j for $j > i$; $1 \le i, j \le b$. For a systematic code, since only the diagonal elements of $\mathbf{S}$ are nonzero, the BFB is selected as per the first b bits of the received word, and then the parity bit is generated. Therefore, the error word of the BFB is either $\mathbf{e} = (0, \cdots, 0, 0)$ or $\mathbf{e} = (0, \cdots, 0, 1)$. As for nonsystematic codes, assume $m_1 = 0$ and generate c_1. If $c_1 \neq r_1$, where r_1 is the first bit of the received word, then setting $m_1 = 1$ will make $c_1 = r_1$. This procedure is successively followed for $m_1, \cdots, m_b$ and then $c_{(b+1)}$ is generated.

B. Next Best Sibling

Let the message word $\mathbf{m}_i$ produce the code word $\mathbf{h}_i$, when the encoder state is zero, $i = 1, 2, \cdots, 2^b$. Let set H consist of all these 2^b code words. The following properties of these code words are well known. P_1: The code word corresponding to the message word $(\mathbf{m}_i \oplus \mathbf{m}_j)$ is $(\mathbf{h}_i \oplus \mathbf{h}_j)$; $\oplus$ being the bitwise modulo-2 operator. P_2: $(\mathbf{h}_i \oplus \mathbf{h}_j) \in H$. P_3: If the encoder state is arbitrary, the code word $\mathbf{c}_i$ corresponding to the message word m_i is given by $\mathbf{c}_i = \mathbf{h}_i \oplus \mathbf{s}$, $\mathbf{s}$ being the code word corresponding to all zero message word applied to the encoder [7].

At the decoder, let $\mathbf{m}_k$ be the message word that produces the code word c_k which is at a minimum distance to the received word r. Therefore, $\mathbf{r} = \mathbf{c}_k \oplus \mathbf{e}$. Using P_3

$$\mathbf{r} = \mathbf{h}_k \oplus \mathbf{s} \oplus \mathbf{e}. \tag{1}$$

The error words of the 2^b forward branches are given by $\mathbf{d}_i = \mathbf{r} \oplus \mathbf{c}_i$, $1 \le i \le 2^b$. Using (1) for $\mathbf{r}$, and P_3 for $\mathbf{c}_i$ gives $\mathbf{d}_i = (\mathbf{h_k} \oplus \mathbf{h_i}) \oplus \mathbf{e}$. Because of P_2, the error words list is therefore given by $(\mathbf{h}_i \oplus \mathbf{e})$, $i = 1, 2, \cdots, 2^b$. In the following, we assume that $h_1, h_2, \cdots$, are in the increasing order of (Hamming) weight.

Consider the case of $\mathbf{e} = (0, \cdots, 0, 0)$. The error words in the increasing order of weight are h_i, $i = 1, 2, \cdots, 2^b$. Therefore, the code words, in the order of distance, are $\mathbf{r} \oplus \mathbf{h}_i$, $i = 1, 2, \cdots, 2^b$. Using (1), $\mathbf{r} \oplus \mathbf{h}_i = (\mathbf{h}_k \oplus \mathbf{h}_i) \oplus s$. By

Paper approved by S. B. Wicker, the Editor for Coding Theory and Techniques of the IEEE Communications Society. Manuscript received April 1, 1996; revised February 27, 1998 and September 15, 1998.

J. D.'Souza was with the Department of Electronics and Electrical Communication Engineering, Indian Institute of Technology, Kharagpur 721302, India. He is now with the Department of Electronics and Communication Engineering, Karnataka Regional Engineering College Surathkal, Mangalore 574157, India.

S. L. Maskara is with the Electronics and Electrical Communication Engineering Department, Indian Institute of Technology, Kharagpur 721302, India.

Publisher Item Identifier S 0090-6778(99)03913-6.

P_1, $(\mathbf{h}_k \oplus \mathbf{h}_i)$ is the code word corresponding to the message word $(\mathbf{m}_k \oplus \mathbf{m}_i)$ when the encoder state is zero. Therefore, by P_3, for $i = 1, 2, \cdots, 2^b$ these message words produce code words in the order of increasing distance. Now consider $c = (0, \cdots, 0, 1)$. The $(b+1)$-tuples $\mathbf{h}_i \oplus \mathbf{e}$, $i = 1, 2, \cdots, 2^b$ may not be in the increasing order of weight. Therefore, we introduce new, unique $(b+1)$-tuples $q_i = h_j$, $1 \le i, j \le 2^b$, such that $\mathbf{q}_i \oplus \mathbf{e}$, $i = 1, 2, \cdots, 2^b$ are in the increasing order of weight. Consequently, the code words in the increasing order of distance are $\mathbf{r} \oplus (\mathbf{q}_i \oplus \mathbf{e})$, $i = 1, 2, \cdots, 2^b$. Proceeding as we did in the case of $c = (0, \cdots, 0, 0)$, the ith code word corresponds to the message word $(\mathbf{m}_k \oplus \mathbf{u}_i)$; u_i is the message word corresponding to the code word $\mathbf{q}_i$.

Consider $\mathbf{e} = (0, \cdots, 0, 0)$. If the metrics corresponding to the weights of $\mathbf{h}_i$, $i = 1, 2, \cdots, 2^b$ are stored in a table M_o, the metric of any sibling can be obtained by a table look-up operation. The message word of the ith branch is $\mathbf{x}_i = \mathbf{m}_k \oplus \mathbf{m}_i$. Similarly, for the $(i+1)$th branch $\mathbf{x}_{i+1} = \mathbf{m}_k \oplus \mathbf{m}_{i+1}$. Or $\mathbf{x}_{i+1} = \mathbf{x}_i \oplus \mathbf{n}_{i+1}$ where $\mathbf{n}_{i+1} = \mathbf{m}_i \oplus \mathbf{m}_{i+1}$. Thus if the b-tuples $\mathbf{n}_j$, $j = 2, \cdots, 2^b$ are stored in a table N_o, then $\mathbf{x}_{i+1}$ can be obtained from x_i through a table look-up operation. Similarly, for $\mathbf{e} = (0, \cdots, 0, 1)$ also, if the metrics corresponding to the weights of $\mathbf{q}_i \oplus \mathbf{e}$, $i = 1, 2, \cdots, 2^b$ are stored in a table M_1 and $\mathbf{v}_{i+1} = \mathbf{u}_i \oplus \mathbf{u}_{i+1}$, $i = 1, 2, \cdots, (2^b - 1)$ are stored in a table N_1, the next best sibling can be generated by table look-up procedure.

III. Comparison Between THR and Punctured Decoding Approaches

The class of THR codes considered in the previous section can be easily transformed to punctured codes, and for rate $b/(b+1)$ the converse is also true [8]. In the following, we therefore make a comparison of the new table-based THR approach and the classical punctured approach for a given code. But such a comparison is somewhat difficult since their computations are not of the same complexity. Traditionally, a computation consists of all operations performed in extending a path by one branch; the main constituent operations are those of accessing the generators and stack reorganization. Since a segment of b branches ending in a two-symbol (TS) node on a punctured code tree, corresponds to a single branch of the THR code tree, for the purpose of comparing, it is convenient to imagine the THR decoder moving on the low rate tree of the equivalent punctured code, b branches per decoding step.

In the punctured approach, if the top node is a TS node, it will be extended up to the next TS node, in b successive steps (or a b-branch computation) of the decoder. The same work will be done in one step by the THR decoder. Although the same number of generators are accessed by both the decoders, the punctured decoder has to perform $b-1$ additional stack reorganizations. On the other hand, a THR computation involves accessing tables and maintaining the requisite tag. But these operations are simpler than those of a stack reorganization. Thus the difference in the complexity of a THR computation and a b-branch computation of a punctured decoder is accounted for by the difference mainly in the number of stack reorganizations and the tag-related operations.

Suppose that a noisy received sequence requires full exploration of the code tree. Now consider a subtree of height b branches, ending in TS nodes. For the THR decoder, only one computation is required to find the BFB of the subtree. All the other $(2^b - 1)$ THR branches of the subtree are explored by table look-up procedure. These siblings are introduced one per decoding step, as a part of a computation that extends a top node. On the other hand, the punctured decoder has to perform, effectively $(2^b - 1)/b$ b-branch computations to explore the full subtree. But our simulation results on average computation (Section V) show that even at $R/R_o = 1$, the computational ratio of the two approaches is close to $1:1$, where R is the code rate and R_o is the computational cutoff rate [1]. This ratio corresponds to that of the noiseless condition. However, since the THR approach is helpful over the noisy segments of received sequence, a slight improvement with channel noise and code rate is also observed. With respect to distribution of computation, as in [6], we have observed a slight superiority of the THR decoder over the punctured decoder. As for stack entries, for a fully explored subtree, both the decoders introduce $(2^b - 1)$ nodes. But our simulation results (Section V) show that even at $R/R_o = 1$, the ratio of the average number of stack entries is $1:b$, which corresponds to that of noiseless condition. Further, the ratio of average decoding time of the two approaches seem to be roughly $2/(b+1):1$, for both noisy and relatively quiet channel conditions. Thus the advantage of THR approach increases with the code rate.

IV. Modified Algorithms

We prove the correctness of the proposed modified algorithms only for the case of punctured codes. But for other codes also, the correctness can be proved using the same philosophy. In the following, the terms father node, sibling of father node, sibling of grandfather node, and depth modulo b, will be denoted by f-node, f-sibling, g-sibling, and depth-mb, respectively.

A. Rate $1/N$ Codes

Let each stack node carry three tags: the W tag to indicate its weight, the S flag to indicate whether sibling of the node is present in the stack, and the L tag to determine whether f-sibling of the node is present in the stack. The steps of the modified algorithm are as follows.

1) Get the top node from the stack. Let its W tag value be x.
2) If sibling is not already present in the stack, then
   Introduce sibling of weight $(N - x)$ into the stack.
   Else if f-sibling is not already present in the stack,
   Introduce f-sibling with weight N into the stack.
3) Keep extending the prior top node (of step 1) along the BFB at each step, until the BFB has nonzero weight y. Siblings are not introduced during these extensions.
4) Set the W tag to y and put the node to the stack. Go to step 1.

As for the L tag, for the initial node in the stack, the tag is set to zero. Whenever a sibling/f-sibling is generated, the L value of the top node is transferred to the L tag of the newly

created sibling/f-sibling, and the L tag of the top node is set equal to its depth value. By this technique, the L tag of a node keeps track of the depth value at which the last sibling was generated along the path to the node. If the L value of a node is less than the depth value of its f-sibling, then the latter is not present in the stack. It should be noted that, for punctured and THR approaches, the details of transferring L tag values between nodes are slightly different from above, but the basic idea is the same.

B. Rate $b/b+1$ Punctured Codes

As in the case of rate $1/N$ codes, let each stack node contain three tags. We assume that the first column of the perforation pattern has two ones. When this is not true, the various cases of depth-mb in the modified algorithm should be labeled accordingly. The steps of the modified algorithm are as follows.

1) Get the top node from the stack.
2) Case 1, depth-$mb = 1$:
   If sibling is not present, introduce sibling of weight 1 into the stack.
   Else if f-sibling is not present, introduce f-sibling of weight 1 into the stack.
   Case 2, depth-$mb = 2$:
   a) If not present, introduce f-sibling of weight 2 into the stack.
   b) If not present, introduce g-sibling of weight 1 into the stack.
   Case 3, depth-$mb = 3, \cdots, b-1, 0$:
   If not present, introduce f-sibling of weight 1 into the stack.
3) Keep extending the prior top node along the BFB at each step, until the BFB has nonzero weight y. Siblings are not introduced during these extensions.
4) Set the W tag to y and put the node to the stack. Go to step 1.

(*Note*: Since for $b = 2$, depth-mb has values of 0 and 1 only, in step 2, case 2 corresponds to depth-$mb = 0$; and case 3 is not applicable.)

Correctness: 1) Step 4 puts a node to stack only if, in step 3, the BFB has a nonzero weight. It is obvious that otherwise a stack reorganization is not required since at step 1 we started with the best node in the stack. At nodes where the emanating branches have only one symbol, the decoder can always find a branch with weight zero. Consequently, at step 4, the node always will have 2 symbols and the weight will be equal to 1. Therefore, in step 2, when depth-$mb = 1$, a sibling of weight 1 is put into the stack if not already present there. In this case, the worse sibling has better metric than f-sibling. Therefore, f-sibling is introduced into the stack only if the top node happens to be the last of the two siblings to reach the stack top.

2) In the case of depth-$mb = 2$, if f-sibling is to be introduced, it will have a weight of 2 (if it had a weight of 1, it would have been introduced at depth-$mb = 1$, when the f-node reached the stack top). Therefore, g-sibling (if it is not present in the stack) which has a weight of 1, will have a better metric than f-sibling. Since the commonly used data structures do not permit us to check if f-sibling is present when g-sibling reaches stack top, we introduce both of them (if they are not present).

3) For depth-$mb \neq 1, 2$ the f-node will have only one symbol and, therefore, the f-sibling will have a weight of 1.

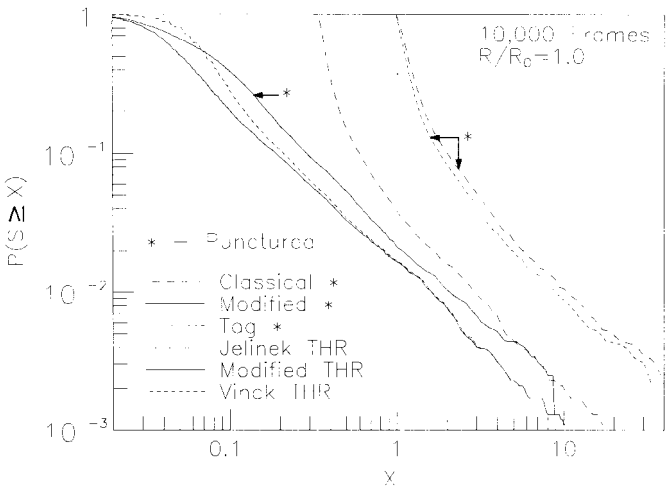

Fig. 1. Distribution of the number of stack entries per decoded bit for punctured and THR decoders. $R = 3/4$.

C. THR Codes

Each stack node is assumed to carry 2 tags: the X tag as index to the requisite tables (Section II), and the L tag. When the BFB has weight zero, let w be the weight of its next best sibling. The steps of the modified algorithm are as follows.

1) Take the top node from the stack.
2) If the top node happens to be the last of the siblings which have a weight $W \leq w$ and if f-sibling is not present in the stack, then introduce f-sibling with weight w.
3) If the next best sibling is available, introduce it into the stack.
4) Keep extending the prior top node (of step 1) along the BFB at each step, until the BFB has nonzero weight. Siblings are not introduced during these extensions.
5) Set the X tag, and place the node in the stack. Go to step 1.

V. Results and Discussions

Simulations have been conducted over hard quantized Gaussian channel, at rates 1/2, 2/3, 3/4, $\cdots$, 7/8 for 3 channel error rates $p/p_o = 1, 1/2, 1/4$ where p_o is the channel error rate corresponding to $R/R_o = 1$. Fig. 1 shows the distribution of the number of stack entries per decoded bit for the different decoders studied, at $R = 3/4$. For the THR approach, the Vinck decoder has a distribution superior to that of the Jelinek decoder. The modified decoder offers improvement over the Vinck decoder for quiet frames, but this difference reduces as the channel becomes noisy. The THR decoders have a far "better" distribution than the classical punctured decoder. The tag-decoder refers to a simplistic modification

TABLE I
NORMALIZED AVERAGES FOR NUMBER OF COMPUTATIONS PER DECODED BRANCH (C_{AV}), NUMBER OF STACK ENTRIES PER DECODED BIT (S_{AV}), AND DECODING TIME PER FRAME (T_{AV}); $*$: INDICATES PUNCTURED

R	Decoder	C_{AV} for $p/p_O =$			S_{AV} for $p/p_O =$			T_{AV} for $p/p_O =$		
		1.0	0.5	0.25	1.0	0.5	0.25	1.0	0.5	0.25
1/2	Tag	1.0000	1.0000	1.0000	0.844	0.962	0.966	0.96	1.01	0.97
	Modified	1.0037	1.0005	1.0001	0.269	0.094	0.044	0.67	0.61	0.52
2/3	Tag*	1.0000	1.0000	1.0000	0.898	0.973	0.991	0.98	1.02	1.07
	Modified*	1.0002	0.9997	1.0000	0.176	0.059	0.026	0.59	0.54	0.49
	Jelinek	0.9870	0.9806	0.9954	0.488	0.490	0.498	0.61	0.63	0.64
	Modified	0.9874	0.9807	0.9954	0.151	0.045	0.021	0.39	0.34	0.32
	Vinck	1.0071	0.9991	1.0059	0.168	0.061	0.031	0.40	0.36	0.32
3/4	Tag*	1.0000	1.0000	1.0000	0.932	0.980	0.993	1.00	1.04	1.03
	Modified*	1.0021	1.0003	1.0001	0.104	0.037	0.016	0.52	0.50	0.47
	Jelinek	0.9743	0.9831	0.9942	0.327	0.328	0.332	0.48	0.46	0.47
	Modified	0.9824	0.9832	0.9942	0.082	0.027	0.012	0.28	0.26	0.24
	Vinck	1.0073	1.0037	1.0056	0.095	0.038	0.018	0.28	0.26	0.25
4/5	Tag*	1.0000	1.0000	1.0000	0.951	0.984	0.995	1.01	1.03	1.05
	Modified*	1.0015	1.0002	1.0001	0.069	0.026	0.011	0.48	0.46	0.50
	Jelinek	0.9829	0.9764	0.9946	0.246	0.244	0.249	0.36	0.37	0.38
	Modified	0.9832	0.9766	0.9947	0.053	0.018	0.009	0.22	0.20	0.19
	Vinck	1.0122	0.9982	1.0068	0.064	0.026	0.013	0.22	0.20	0.20
5/6	Tag*	1.0000	1.0000	1.0000	0.963	0.988	0.995	1.02	1.04	1.05
	Modified*	1.0014	1.0002	1.0001	0.050	0.019	0.009	0.46	0.46	0.45
	Jelinek	0.9722	0.9809	0.9918	0.195	0.197	0.199	0.30	0.31	0.32
	Modified	0.9729	0.9810	0.9918	0.036	0.013	0.007	0.18	0.17	0.17
	Vinck	1.0042	1.0027	1.0042	0.045	0.020	0.010	0.18	0.17	0.17
6/7	Tag*	1.0000	1.0000	1.0000	0.970	0.989	0.996	1.03	1.04	1.06
	Modified*	1.0013	1.0003	1.0001	0.039	0.016	0.008	0.46	0.45	0.45
	Jelinek	0.9470	0.9770	0.9920	0.158	0.163	0.166	0.27	0.29	0.30
	Modified	0.9471	0.9771	0.9921	0.025	0.011	0.006	0.18	0.17	0.17
	Vinck	0.9805	1.0007	1.0048	0.033	0.016	0.009	0.17	0.18	0.17
7/8	Tag*	1.0000	1.0000	1.0000	0.976	0.992	0.997	1.03	1.04	1.06
	Modified*	1.0009	1.0001	1.0000	0.031	0.012	0.006	0.44	0.44	0.44
	Jelinek	0.9728	0.9798	0.9945	0.139	0.141	0.143	0.25	0.25	0.25
	Modified	0.9727	0.9797	0.9943	0.022	0.008	0.005	0.16	0.15	0.15
	Vinck	1.0086	1.0016	1.0059	0.028	0.012	0.007	0.15	0.15	0.14

of classical punctured decoder, wherein with the help of a tag, the introduction of worse sibling is delayed until the better metric sibling reaches the stack top [2]. It is seen to give only a slight improvement. But the modified punctured decoder gives at least as much improvement as the Jelinek THR decoder. The average values for computations, stack entries, and decoding time for different decoders are given in Table I, normalized with respect to those of classical punctured decoder. The following observations can be made. First, with respect to computation, the THR decoders have only a slight edge over the punctured approach; but with respect to stack entries and decoding time, their performance is superior. Second, the proposed modifications give improvement with respect to both stack entries and decoding time. Third, the first two improvements increase with the code rate. Fourth, while the decoding speeds of modified THR and Vinck decoders are comparable, the former is slightly better with respect to computation and stack entries. But our Vinck decoder was implemented using tables and tags, since with respect to decoding speed, such an implementation is more efficient than that in [4]. We may also note that the modified THR decoder has the desirable feature that, irrespective of the code rate, a decoding step introduces only one sibling into the stack. Simulation results thus indicate that the proposed techniques enable low complexity stack decoding.

ACKNOWLEDGMENT

The authors are grateful to the reviewers and the editor for the review of this communication.

REFERENCES

[1] S. Lin and D. J. Costello, Jr., *Error Control Coding.* Englewood Cliffs, NJ: Prentice-Hall, 1983.
[2] F. Jelinek, "A fast sequential decoding algorithm using a stack," *IBM J. Res. Develop.*, vol. 13, pp. 675–685, Nov. 1969.
[3] A. J. Vinck, A. J. P. Paepe, and A. P. M. Schalkwijk, "A class of binary rate one-half convolutional codes that allows an improved stack decoder," *IEEE Trans. Inform. Theory*, vol. IT-26, pp. 389–392, July 1980.

[4] A. J. Vinck, "A low complexity stack decoder for a class of binary rate $(n-1)/n$ convolutional codes," *IEEE Trans. Commun.*, vol. COM-32, pp. 476–479, Apr. 1984.
[5] J. B. Cain, G. C. Clark, and J. Geist, "Punctured convolutional codes of rate $(n-1)/n$ and simplified maximum likelihood decoding," *IEEE Trans. Inform. Theory*, vol. IT-25, pp. 97–100, Jan. 1979.
[6] G. Bégin and D. Haccoun, "Performance of sequential decoding of high-rate punctured convolutional codes," *IEEE Trans. Commun.*, vol. COM-42, pp. 966–978, Feb./Mar./Apr. 1994.
[7] W. H. Ng, "Study on decoding recovery behavior for convolutional codes," *IEEE Trans. Inform. Theory*, vol. IT-16, pp. 795–797, Nov. 1970.
[8] J. D'Souza and S. L. Maskara, "Simple method for constructing equivalent punctured codes for given true high rate codes," *Electron. Lett.*, vol. 30, pp. 24–26, Jan. 1994.