AN OVERVIEW OF DECODING PROCEDURES FOR TRELLIS CODES

Vojin Šenk, Predrag Radivojac, and Ivan Stanojević
University of Novi Sad, School of Engineering

(*invited paper*)

I. INTRODUCTION

Although most of the early work in coding included algebraic code constructions and fast decoding methods, latest trends definitely favor probabilistic i.e. trellis codes with soft-decision decoding algorithms. Recent emerge of parallel concatenated convolutional (so-called turbo) codes together with MAP iterative decoding achieved performance very close to the channel capacity.

Trellis codes are still dominant in applications to deep space and satellite communications where power is very expensive and bandwidth abundant, spectrally efficient applications to data transmission (TCM) over narrowband channels, digital audio broadcasting, and mobile systems (both TDMA and CDMA) of the next generation [8]. On the other hand, block codes still prevail in all types of data storage systems due to their simplicity, small buffering requirements, and high speed and file transfer protocols where hard-decision decoding provides natural form of ARQ based protocols. It seems that turbo codes will soon penetrate into these isolated areas.

Algorithms for decoding trellis codes are thus an important and active area of research from the very beginning until now. Although the optimal codeword-oriented (Viterbi) and symbol oriented (BCJR) algorithms are well-known and frequently used, the quest for suboptimal algorithms whose complexity is greatly reduced and probability of error not significantly increased yields new efficient procedures almost regularly.

Trellis decoding algorithms have found a wide application in different areas outside the strict region of decoding trellis (including turbo) codes. Block (including lattice) codes are very efficiently decoded using trellis search procedures, maximum-likelihood sequence estimation (as in ISI equalization) is also performed using trellis-search techniques. Trellis search is used for data compression with a fidelity criterion, for pattern recognition, as well as in many other applications.

This paper will try to give a systematization of these algorithms, from the well-known to some that are less known or entirely new and presented here for the first time.

II. DEFINITION OF TRELLIS CODES

A trellis encoder can be simply described as a Mealy's finite-state machine whose structure may be displayed with the aid of a graph, tree or trellis diagram. The trellis encoder maps sequences of K-dimensional q-ary alphabet input vectors into sequences of N-dimensional r-ary alphabet output vectors. Although the encoder is generally nonlinear and time-varying, virtually all trellis encoders of practical interest are linear time-invariant (also called fixed convolutional encoders – FCE), for which $q = r$. Mapping of an FCE, $F(\cdot)$, satisfies the following conditions:

$$\begin{aligned} &F(a \cdot \mathbf{i}_{[0,\infty)}) = a \cdot F(\mathbf{i}_{[0,\infty)}) \\ &F(\mathbf{i}'_{[0,\infty)} + \mathbf{i}''_{[0,\infty)}) = F(\mathbf{i}'_{[0,\infty)}) + F(\mathbf{i}''_{[0,\infty)}) \end{aligned} \tag{1}$$

where a belongs to the input/output alphabet, $\mathbf{i}_{[0,\infty)}$ is any input sequence and $F(\mathbf{i}_{[0,\infty)})$ is the corresponding output sequence. It is assumed that input and output symbols belong to some finite field. Also, for any $d > 0$, if $\mathbf{x}_{[0,\infty)} = F(\mathbf{i}_{[0,\infty)})$ and $\mathbf{i}'_l = \mathbf{i}_{l-d}$, $\mathbf{i}'_{[0,d)} = \mathbf{0}_{[0,d)}$ then $F(\mathbf{i}'_{[0,\infty)}) = \mathbf{x}'_{[0,\infty)}$, where $\mathbf{x}'_l = \mathbf{x}_{l-d}$, $\mathbf{x}'_{[0,d)} = \mathbf{0}_{[0,d)}$. It is easily verified that $F(\cdot)$ can be represented by the convolution, i.e., if $\mathbf{i}_l = (i_l^{(1)}, i_l^{(2)}, \ldots, i_l^{(K)})$ and $\mathbf{x}_l = (x_l^{(1)}, x_l^{(2)}, \ldots, x_l^{(N)})$, then

$$x_l^{(n)} = \sum_{k=1}^{K} \sum_{m=0}^{\infty} i_{l-m}^{(k)} \cdot g_m^{(k,n)}, \quad n \in \{1,2,\ldots,N\}. \tag{2}$$

where $g_{[0,\infty)}^{(k,n)}$, $k \in \{1,2,\ldots,K\}$, $n \in \{1,2,\ldots,N\}$ are generator sequences. If all $g_{[0,\infty)}^{(k,n)}$ are finite, i.e. there exists some $0 \le s < \infty$, such that $g_{(s,\infty)}^{(k,n)} = 0_{(s,\infty)}$ for all k, n, then the smallest such value s, denoted $\mathcal{M}$, is called the code memory and the encoder is of feedforward type.

Let us now consider a feedforward convolutional encoder with memory length $\mathcal{M}$. At any time instant (depth or level) l, the encoder output N-tuple $\mathbf{x}_l$ depends on the current input K-tuple $\mathbf{i}_l$, and $\mathcal{M}$ previous inputs $\mathbf{i}_{l-1}, \ldots, \mathbf{i}_{l-\mathcal{M}}$. The overall functioning of the encoder can be mapped on a trellis diagram, whereon a *node* represents one of $q^{K\mathcal{M}}$ encoder states, while a *branch* connecting two nodes represents the encoder output associated to the transition between the corresponding encoder states.

A trellis, which is a visualization of the state transition diagram with a time element incorporated, is characterized by q^K branches stemming from and entering each state, except in the first and last $\mathcal{M}$ branches (respectively called head and tail of the trellis). The branches at the l-th time instant are labeled by sequences $\mathbf{x}_l \in \mathcal{X}^N$. A sequence of l information K-tuples, $\mathbf{i}_{[0,l)}$ specifies a *path* from the root node to a node at the l-th level and, in turn, this path specifies a *codeword* $\mathbf{x}_{[0,l)} = \mathbf{x}_0 * \mathbf{x}_1 * \ldots * \mathbf{x}_{l-1}$, where $*$ denotes a concatenation of two sequences. The *code rate* is defined as $R = (K/N) \cdot \log_2(q)$.

An overwhelming majority of today's digital communication forms incorporate transfer of information separated in frames of some length. Here, we consider framed data, where the length of each information frame equals L branches (thus $K \cdot L$ q-ary symbols) and the length of the coded frame is $L+\mathcal{M}$ branches ($N \cdot (L+\mathcal{M})$ r-ary symbols), where the $\mathcal{M}$ known K-tuples (usually all zeros) are added at the end of the sequence to force encoder into the desired terminal state. It is said that such codes suffer a fractional rate loss giving overall code

rate $R = (L/(L + \mathcal{M}))\cdot(K/N)\cdot \log_2(q)$ information bits per channel symbol. Clearly, this rate loss has no asymptotic significance.

In this overview we restrict the discussion to binary fixed trellis channel codes for which $r = q = 2$. Any further generalization will be straightforward.

III. Discrete Memoryless Channels and Decoding Metric

In the sequel we further restrict our discussion to transmission over a discrete memoryless channel (DMC) that can be provided by equalization that is incorporated into the waveform channel. Also, we suppose there is no feedback from the receiver to the transmitter. The term discrete memoryless channel means that

$$P[y_n \mid x_0,\ldots,x_{n-1},x_n,y_0,\ldots,y_{n-1}] = P[y_n \mid x_n], \tag{3}$$

and

$$P[y_1,\ldots,y_N \mid x_1,\ldots,x_N] = \prod_{n=1}^{N} P[y_n \mid x_n]. \tag{4}$$

The DMC arises when the waveform signal is exposed to additive white Gaussian noise and then sampled and quantized to enable digital processing. If quantization is binary, the quantizer is generally called a hard quantizer, in contrast to multilevel quantizer which is called a soft quantizer. The resulting hard-quantized output channel is the binary symmetric channel (BSC) with channel error (or crossover) probability $p \in [0, ½]$.

The task of a decoder which minimizes the sequence error probability is to find a sequence which maximizes the joint probability of input and output channel sequences

$$P[\mathbf{y}_{[0,L+\mathcal{M})},\mathbf{x}_{[0,L+\mathcal{M})}] = P[\mathbf{y}_{[0,L+\mathcal{M})} \mid \mathbf{x}_{[0,L+\mathcal{M})}]\cdot P[\mathbf{x}_{[0,L+\mathcal{M})}]. \tag{5}$$

Since a source and secrecy coding that come before channel coding usually set all probabilities $P[\mathbf{x}_{[0,L+\mathcal{M})}]$ to be equal, it is sufficient to find a procedure that maximizes $P[\mathbf{y}_{[0,L+\mathcal{M})} \mid \mathbf{x}_{[0,L+\mathcal{M})}]$, and a decoder that always chooses as its estimate one of the sequences that maximize it or

$$\begin{aligned}\mu(\mathbf{y}_{[0,L+\mathcal{M})} \mid \mathbf{x}_{[0,L+\mathcal{M})}) &= A\cdot(\log_2 P[\mathbf{y}_{[0,L+\mathcal{M})} \mid \mathbf{x}_{[0,L+\mathcal{M})}] \\ &- f(\mathbf{y}_{[0,L+\mathcal{M})}) = A\cdot\sum_{l=0}^{L+\mathcal{M}}(P[\mathbf{y}_l \mid \mathbf{x}_l] - f(\mathbf{y}_l))\end{aligned}, \tag{6}$$

(where A is a suitably chosen constant, and $f(\cdot)$ any function) is called a maximum-likelihood decoder (MLD). This expression is called a metric. This type of metric suffers one significant disadvantage because it is suited only for comparison between paths of the same length. Some algorithms, however, employ a strategy of comparing paths of different length or assessing likelihood of such paths with the aid of some thresholds. The metric that enables comparison for this type of algorithms is called the Fano metric. It is defined as

$$\begin{aligned}\mu_F(\mathbf{y}_{[0,l)} \mid \mathbf{x}_{[0,l)}) &= A\cdot\log_2\frac{P[\mathbf{y}_{[0,l)},\mathbf{x}_{[0,l)}]}{P[\mathbf{y}_{[0,l)}]} \\ &= A\cdot\sum_{n=0}^{l\cdot N}(\log_2\frac{P[y_n \mid x_n]}{P[y_n]} - R)\end{aligned} \tag{7}$$

IV. Decoding Trellis Codes

In this paper our primary goal is to classify and analyze trellis decoding algorithms. As stated above, the aim of the search procedure is to find a path with the highest possible likelihood i.e. metric. There are several possible classifications of decoding procedures. According to the decoder's strategy in extending the most promising path candidates we systematize them into breadth-first, metric-first, depth-first algorithms and into sorting and nonsorting if the procedure performs any kind of path comparison (sifting or sorting) or not. Moreover, decoding algorithms can be classified into searches that minimize the sequence or symbol error rate.

The usual measure of algorithm efficiency in channel coding is its complexity (arithmetic and storage) for a given probability of error. In the strict sense arithmetic or computational complexity is the number of arithmetic operations per decoded symbol, branch, or frame. However, it is a usual practice to track only the number of node computations, which makes sense because all such computations require approximately the same number of basic machine instructions. A node computation (or simply computation) is defined as the total number of nodes extended (sometimes it is the number of metrics computed, which is 2^K times greater) per decoded branch or information frame $\mathbf{i}_{[0,L+\mathcal{M})}$. One single computation consists of determining the state in which the node is and computing the metrics of all its successors. For most practical applications with finite frame length it is usually sufficient to observe node computations since a good prediction of search duration can be precisely predicted. Nevertheless, for asymptotic behavior it is necessary to track the sorting requirements too. Another important aspect of complexity is storage (memory or space) which is the amount of auxiliary storage that is required for decoding (memory, processors working in parallel etc.). Thus, space complexity of an algorithm is the size (or number) of resources that must be reserved for its use, while the computational or more precisely time complexity reflects the number of accesses to this resources taking into account that any two operations done in parallel by the spatially separated processors should be counted as one. The product of these two, the time-space complexity is possibly the best measure of the algorithm cost for it is insensitive to time-space tradeoff such as parallelization or the use of precomputed tables, although it also makes sense to keep the separate track of these two. Finally, for selecting which algorithm to use one must consider additional details that we omit here but which can sometimes cause unexpected overall performance or complicate the design of a real-time decoder. They include complexity of the required data structure, buffering needs and applicability to available hardware components.

A. Maximum-Likelihood Decoding

The Viterbi Algorithm

The Viterbi algorithm (VA) is an optimal decoding algorithm in the sense that it always finds the nearest path to the noisy modification of the encoder output sequence $\mathbf{x}_{[0,L+\mathcal{M})}$, and it is quite useful when the code has a short memory. The key to Viterbi (maximum-likelihood) decoding lies in the Principle of Non-Optimality [4]: If the paths $\mathbf{i}'_{[0,l)}$ and $\mathbf{i}''_{[0,l)}$ terminate at the same state of the trellis and

$$\mu(\mathbf{y}_{[0,l)},\mathbf{x}'_{[0,l)}) > \mu(\mathbf{y}_{[0,l)},\mathbf{x}''_{[0,l)}), \tag{8}$$

then $\mathbf{i}''_{[0,l)}$ cannot be the first l branches of one of the paths $\mathbf{i}_{[0,L+\mathcal{M})}$ that maximize (gore). This principle which some authors call the Principle of Optimality literally specifies the most efficient MLD procedure for decoding trellis codes.

Software realization of the Viterbi algorithm is a classical application of dynamic programming. Structurally, the algo-

rithm contains $2^{K\mathcal{M}}$ lists, one for each state, where the paths whose states correspond to the label indices are stored, compared, and the best one of them retained. The algorithm can be described recursively as follows

1. *Initial Condition:* Initialize the starting list with the root node and set its metric to zero.
2. *Path Extension:* Extend all the paths (nodes) by one branch to yield new candidates (there is only one successor for each $l \geq L$) and classify these candidates into corresponding $2^{K\mathcal{M}}$ lists (or less for $l < \mathcal{M}$ and $l > L$). Each list (again, except in the head and tail of the trellis) contains 2^K paths.
3. *Path Selection:* From each list at depth l, a path $\mathbf{x}_{[0,\, l)}$ with the largest metric is selected for the next step, and the others discarded. If two or more paths have the same metric, i.e. if they are equally likely, choose the best one at random. If $l = L + \mathcal{M}$ take the only survivor from its list and transfer the corresponding information sequence to the output; otherwise go to step 2.

Consider now the amount of "processing" done at each depth l, $\mathcal{M} < l \leq L$, where all of the $2^{K\mathcal{M}}$ states of the trellis code are present. For each state it is necessary to compare 2^K paths that merge in that state, discard all but the best path, and then compute and send the metrics of 2^K of its successors to the depth $l + 1$.

Consequently, the computational complexity of the VA exponentially increases with $\mathcal{M}$. These operations can be easily parallelized, but then their number rises as its number of node computations decreases. The total time-space complexity of the algorithm increases exponentially with the memory length.

B. Sequential Decoding

Since the total time-space complexity of the VA exponentially grows with the code memory length this is a serious limitation to achieving very low error rates that exponentially decrease with $\mathcal{M}$. Therefore, it is essential to find a decoding scheme whose error-probability exponentially decreases with $\mathcal{M}$, but with computational and space complexity linearly dependent on it. Although this scheme is not completely feasible, there is a number of algorithms that are close to it. All these techniques that in fact search only the high-probability paths through a trellis (or tree) are known as sequential decoding and employ breadth-first, metric-first, or depth-first search strategy.

Breadth-first Algorithms

a) The M-algorithm

Since most survivors in the VA usually possess much smaller metrics than does the best one, all the states or nodes kept are not equally important. It is intuitively reasonable to assume that unpromising survivors can be omitted with a negligible probability of discarding the best one. The M-algorithm [3] is one such modification of the Viterbi algorithm; all candidates are stored in a single list and the best $M \leq 2^{K\mathcal{M}}$ survivors are selected from the list in each cycle. The steps of the M-algorithm are:

1. *Initial Condition:* Initialize the list with the root node and set its metric to zero.
2. *Path Extension:* Extend all the paths of length l by one branch and classify all contenders (paths of length $l + 1$) into the list. If two or more paths enter the same state keep the best one.
3. *Path Selection:* From the remaining paths find the best M candidates and delete the others. If $l = L + \mathcal{M}$ take the only survivor and transfer its corresponding information sequence to the output; otherwise go to step 2.

Defined in this way, the M-algorithm performs trellis search, while, when the state comparison in step 2 is omitted, it searches the tree, saving much time on comparisons but with slightly increased error probability. When applied to decoding infinitely long sequences, it is usual that comparisons performed in step 2 are substituted with the so-called ambiguity check [3] and a release of one decoded branch. In each step this algorithm performs M node computations, and employing any sifting procedure (since the paths need not be sorted) perform $\sim M \cdot 2^K$ metric comparisons. If performed, the Viterbi-type discarding of step 2 requests $\sim M^2 \cdot 2^K$ state and metric comparisons. This type of discarding can be performed with $\sim M \cdot \log_2 M$ comparisons (or even linearly) but than additional storage must be provided (in the letter case it grows exponentially with $\mathcal{M}$). The space complexity grows linearly with the information frame length L and parameter M.

b) The Generalized Viterbi Algorithm

In contrast to the Viterbi algorithm which is a multiple-list single survivor algorithm, the M-algorithm is a single-list multiple-survivor algorithm. The natural generalization to a multiple-list multiple-survivor algorithm was first suggested by Hashimoto [2]. Since all the lists are not equally important, this algorithm, originally called the generalized Viterbi algorithm (GVA), utilizes only $2^{\mathcal{M}_1}$ lists (labels), where $\mathcal{M}_1 \leq \mathcal{M}$. In each list from all $2^{K(\mathcal{M}-\mathcal{M}_1+1)}$ paths it retains the best M_1 candidates. The algorithm can be described as follows.

1. *Initial Condition:* Initialize the starting label with the root node and set its metric to zero.
2. *Path Extension:* Extend all the paths from each label by one branch and classify all successors into the appropriate label. If two or more paths enter the same state keep the best one.
3. *Path Selection:* From the remaining paths of each label find the best M_1 and delete the others. If $l = L + \mathcal{M}$ take the only survivor and transfer its information sequence to the output; otherwise go to the step 2.

When $\mathcal{M}_1 = \mathcal{M}$, and $M_1 = 1$ the GVA reduces to the Viterbi algorithm, and for $\mathcal{M}_1 = 0$, $M_1 = M$ it reduces to the M algorithm. Like the M-algorithm GVA in each step performs M_1 node computations per label, and employing any sifting procedure (since the paths need not be sorted) performs $\sim M_1 \cdot 2^K$ metric comparisons. If performed, the Viterbi-type discarding of step 2 requests $\sim M_1^2 2^K$ or less state and metric comparisons per label.

c) The T-algorithm

Another breadth-first algorithm, popularly called the T-algorithm, was suggested by Simmons [1]. Its steps are:

1. *Initial Condition:* Initialize the list with the root node and set its metric to zero.
2. *Path Extension:* Extend all the paths of length l by one branch and classify all contenders (paths of length $l + 1$)

into the list. If two or more paths enter the same state keep the best one.

3. *Path Selection:* From the remaining paths find the best one and discard all paths whose metric satisfies $\mu_{path} \le \mu_{best\ path} - T$, where T is a parameter. If $l = L + \mathcal{M}$ take the only survivor and transfer its information sequence to the output; otherwise go to step 2.

For this algorithm it is important to consider one more detail. Since the number of survivors depends on channel noise, it is therefore a random variable and may grow quite large. Therefore, a limit must be enforced on the list size, say M, in order to enable practical implementation. Simulations and comparative analysis show that for the same error-rate, the T-algorithm yields lower average computational effort than the M-algorithm [1] [7], but buffering requirements are increased. This is a consequence of the variable list size that enables better recovery of the correct path when the channel noise is relatively high. Additionally, the T-algorithm saves much time on sifting since the path elimination can be performed in only 2 passes through the list.

d) The Generalized Viterbi-T algorithm

One more variation of the breadth-first procedures, generalized Viterbi-T algorithm is presented in [7]. The algorithm is similar to the GVA, but instead of retaining best M_1 paths, in each label it retains those paths whose metric is $\mu_{path} \le \mu_{best\ path} - T_1$, where T_1 is a suitably chosen threshold. Similarly to the comparison in c) this algorithm is equivalent to the VA for $\mathcal{M}_1 = \mathcal{M}$ and $T \to \infty$, and to the T-algorithm for $\mathcal{M}_1 = 0$ and $T_1 = T$. The performance of the GVTA was tested on BSC, and simulations showed [7] that it reaches similar number of node computations as the GVA but with reduced sorting effort.

Metric-first Algorithms

Metric-first and depth-first sequential decoding is a name for a class of algorithms that compare paths according to their Fano metric (one against another or with some thresholds) and on that basis decide which node to extend next, which to delete in metric first procedures or whether to proceed with current branch or go back. These algorithms generally extend fewer nodes for the same performance, but have increased sorting requirements.

Sequential decoding algorithms have a variable computation characteristic which results in large buffering requirements, and occasionally large decoding delays and/or incomplete decoding of the received sequence. Sometimes, when almost error-free communication is required or when retransmission is possible, this variable decoding effort can be an advantage. For example, when a decoder encounters an excessive number of computations, it indicates that a frame is possibly very corrupted meaning that the communication is insufficiently reliable and can ultimately cause error patterns in decoded sequence. In such situations the decoder gives up decoding and simply requests retransmission. These situations are commonly called erasures, and decoding incomplete. A complete decoder such as the Viterbi decoder would be forced to make an estimate, which may be wrong. The probability of buffer overflow is several orders of magnitude larger than the probability of incorrect decision when the decoder operates close to the computational cutoff rate.

The performance of sequential decoding has traditionally been evaluated in terms of three characteristics: the probability of sequence error, the probability of failure (erasure), and the Pareto exponent associated with decoding effort.

a) The Stack Algorithm

The stack (or ZJ) algorithm was for the first time suggested by Zigangirov [6] and later independently by Jelinek [6]. As its name indicates, the algorithm contains a stack (in fact, a list) of already searched paths of varying lengths, ordered according to their metric values. At each step, the path at the top of the stack (the best one) is replaced by its 2^K successors extended by one branch, with correspondingly augmented metrics. The check whether two or more paths are in the same state is not performed. This algorithm has its numerous variations and we first consider the basic version that is closest to Zigangirov's:

1. *Initial Condition:* Initialize the stack with the root node and set its Fano metric to zero (or some large positive number to avoid arithmetic with negative numbers, but low enough to avoid overflow).
2. *Path Extension:* Extend the best path from the stack by one branch, delete it, sort all successors, and then merge them with the stack so that it is ordered according to the path metrics.
3. *Path Selection:* Retain the best Z paths according to the Fano metric. If the top path has the length $l = L + \mathcal{M}$ branches, transfer its information sequence to the output; otherwise go to step 2.

It is obvious that this algorithm does not consider path merging since the probability that the paths of the same depth and the same state can be stored in the stack is rather small. Nonetheless, some authors [6] propose that a following action should be added to the step 2

2a. If any of the 2^K new paths merges with a path already in the stack, keep the one with the higher metric.

The stack algorithm is based on the Non-Selection Principle [4]: If the paths $\mathbf{i}'_{[0,L+\mathcal{M})}$ and $\mathbf{i}''_{[0,L+\mathcal{M})}$ through the tree diverge at depth j and

$$\min\left\{\mu(\mathbf{x}'_{[0,l)}, \mathbf{y}_{[0,l)})\right\}_{l\in[j+1,L+\mathcal{M})} > \min\left\{\mu(\mathbf{x}''_{[0,l)}, \mathbf{y}_{[0,l)})\right\}_{l\in[j+1,L+\mathcal{M})} \tag{9}$$

then $\mathbf{i}''_{[0,L+\mathcal{M})}$ cannot be the path at the top of the stack when the stack algorithm stops.

The computational complexity of the stack algorithm is almost unaffected by the code memory length but well depends on the channel performance. Its computational complexity is a random variable and so is its stack size if not otherwise limited. The upper bound on the computational complexity is given by

$$P[C \ge \eta] < A \cdot \eta^{-\rho} \qquad 0 < \rho \le 1, \tag{10}$$

where A is a constant and ρ is a power that goes to unity as $R \to R_0 < R_C$ and to zero as $R \to R_C$, where R_C is the channel capacity and R_0 is called cutoff rate [4]. The distribution described in (10) is called a Pareto distribution, and ρ is called a Pareto exponent. It also holds that for time-varying trellis codes ρ is upper bounded by ∞.

b) The Haccoun-Ferguson's Algorithm

Haccoun and Ferguson [17] generalized the stack algorithm in order to trade error probability with decreased erasure probability. Instead of extending one node before sorting, their algorithm extends several most promising candidates thus having increased number of node computations when SNR in channel is high, but reducing it for low SNR-s. Šenk and Radivojac [18] further generalized this procedure since each following node extension can be adjusted to a different type of channel.

c) The Multiple Stack Algorithm

The multiple stack algorithm is an example of a metric-first and depth-first search. It is designed to exploit the good characteristics of the basic stack algorithm with decreased erasure probability. This algorithm performs a metric-first search with a smaller initial stack than the original stack algorithm. If no path reaches decision depth, a small number of paths are taken off the stack and placed into another stack of smaller size; if the second search is unsuccessful it generates a new search in the same way. This procedure continues until some path reaches the decision depth or limit on node computations, C_{lim}, is violated. The steps of the algorithm are

1. Obtain the root node, set its Fano metric to zero, and place it into the initial stack.
2. Execute a standard stack algorithm until the present stack is full. If the total stacks exceed a limit release as output the path in the best path buffer and stop. If a path reaches depth $L + \mathcal{M}$, go to step 4.
3. Make a new stack with the best T paths from the previous one, and go to step 2.
4. If a path that reached the decision depth is in the initial stack, release it as output and stop. Else proceed to step 5.
5. Check whether a path is the best so far found at depth $L + \mathcal{M}$ and store it as the decoder's tentative decision buffer if it is. Drop the present stack, return to the previous one and go to step 2.

The advantage of the MSA over the conventional stack algorithm arises from the use of additional stacks. The initial stack is made large enough that only very noisy channel sequences require the use of the additional stacks. In such cases instead of extensively exploring the code tree, the algorithm quickly moves through it and finds a reasonable tentative decision. It then drops the current stack and goes back to explore previous stacks in detail searching for the most likely path. This is provided by the size of auxiliary stacks that is substantially lower than the initial stack size Z_{init}. If the limit on node computations is not small, it is highly likely that the algorithm will make at least one tentative decision, thus decreasing the erasure probability.

When the channel sequence does not contain a large amount of errors the complexity of the algorithm is the same as that of the original stack algorithm. On the other hand, when it encounters a very noisy sequence it reduces the search, but it highly depends on the algorithm parameters: C_{lim} and the size of auxiliary stacks. The extensive simulation of the MSA is performed in [12], and suggests the use of the MSA for channels with high signal to noise ratio. This algorithm can be also used for the channels with intersymbol interference, e.g. magnetic recording channel. All the modifications of the original stack algorithm that reduce the sorting cost can be applied here as well.

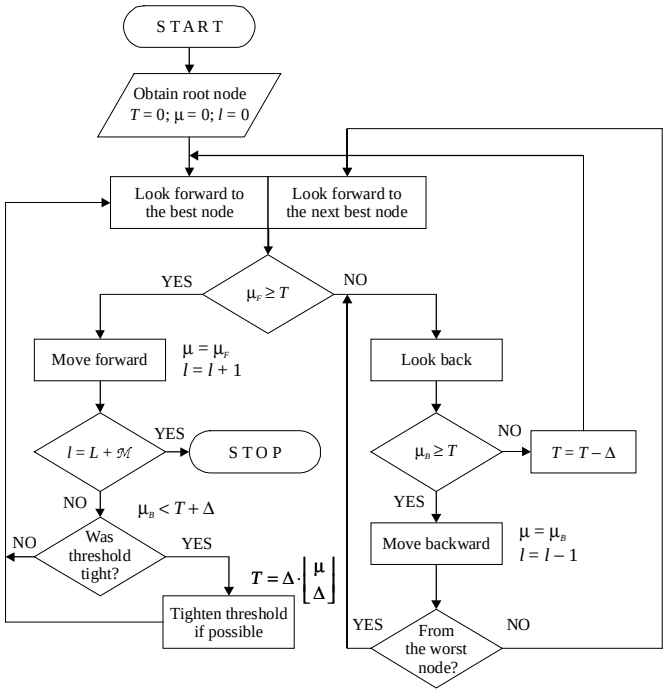

Fig. 1. The Fano algorithm - flowchart

Depth-first algorithms

a) The Fano algorithm

The Fano algorithm [11] is one of the most practical algorithms for implementation. Its basic feature is that it examines only one path at a time and proceeds along it as long as its metric grows. When its metric begins to decrease it backtracks along the path and explores other extensions stemming from it. It accomplishes this by varying a comparison threshold in steps of magnitude Δ or its multiples. Since the algorithm expects metric always to grow, the threshold T is tightened (increased by the largest multiple of Δ such that the metric of the path does not violate the threshold) whenever the metric is growing sufficiently on a forward search. When the algorithm encounters a metric dip, it goes backward and the metric is relaxed (lowered by Δ). No node is visited twice with the same threshold value. In each subsequent visit of the same node, the threshold must be lower than when it was previously searched, which prevents it from being in a loop forever. It is assumed that there is a fictitious node, backwards from the root node, with metric $-\infty$, so that a look back from the root node always results in lowering of the threshold. The flowchart of the algorithm is shown in Fig. 1.

The Fano algorithm has very low storage and sorting requirements. It trades them for a large number of node computations since it has to visit some nodes many times as the channel worsens. The extensive research showed that the Fano algorithm has practically the same performance as the stack algorithm.

Some other depth-first procedures of less importance, e.g. the single stack algorithm or the 2-cycle algorithm can be found in [3].

Bidirectional Algorithms

Another class of decoding algorithms are those that exploit bidirectional decoding which is designed for framed data. Almost all unidirectional procedures have their bidirec-

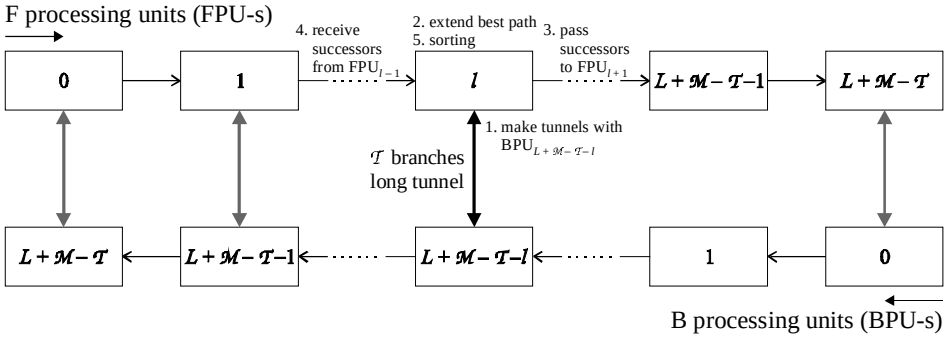

Fig. 2. Organization of the new bidirectional decoder

tional supplements since Forney showed that decoding can start from the end of the sequence provided that the trellis contains a tail. All bidirectional algorithms employ two searches from both sides. The forward search is performed using the original trellis code while the backward one employs the *reverse code*. The reverse trellis code is obtained from the original code by time reversing.

a) The Bidirectional Stack Algorithm

This algorithm is independently proposed by Šenk and Radivojac [5][15][16], and Kallel and Li [14]. It uses two stacks: F (forward) and B (backward, that uses the reverse code). It is based on notions of *tunnel*, *tentative decision* and *discarding criteria*. The tunnel is the unique sequence $\mathcal{T}$ ($0 \le \mathcal{T} \le \mathcal{M}$) branches long that connects two states in the trellis. The tentative decision is the sequence $L + \mathcal{M}$ branches long that connects the known initial and terminal trellis states (direction does not matter here) that has the highest accumulated metric of all the sequences of that length analyzed so far. A set of discarding criteria is a means to tell beforehand whether a partly explored path is likely to be a part of the finally decoded sequence or not (in the latter case, the path may be eliminated from the subsequent search). Since the second version of the algorithm is a special case of [15] (when $\mathcal{T} = 0$) we give the steps of the BSA as:

1. Place the root node into F stack, and the unique terminal node into B stack, associating them the zero metric. Make one of this stacks active (e.g. the F one).
2. Choose the node with the largest metric (of length, say, l) from the active stack and eliminate it from the stack. Link it via a tunnel (if a tunnel is possible, i.e. if the states match) to each of the existing paths in the other stack whose lengths are $L - l + \mathcal{M} - \mathcal{T}$ (if a tunnel is $\mathcal{M}$ branches long, then the best path from the active stack can be linked to all the paths from the other stack whose lengths are $L - l$). The total length of the paths obtained in this way is $l + \mathcal{T} + (L - l + \mathcal{M} - \mathcal{T}) = L + \mathcal{M}$ branches. Store the best one into the tentative decision register. If there is already a path in the register keep the better. Prune the paths remaining in both stacks according to any of discarding criteria used. If both stacks are emptied in this way, output the tentative decision as the decoder's final decision and terminate the algorithm. Otherwise, evaluate the metrics of all the successors of the processed path, and eliminate all of them that do not conform to the discarding criteria established.
3. Sort the remaining successors into the active stack according to their metrics applying any tie-breaking rule. Change the active stack and return to step 2.

After each tentative decision, several discarding criteria can be applied. In [15] Šenk and Radivojac applied the Non-Selection Principle and the maximum-likelihood criterion described. The algorithm can be easily performed by two processors, although one node computation lasts longer then in the original stack algorithm. Simulations showed [15] that the Pareto exponent of the BSA in the moment when the final decision is obtained is approximately doubled, but the discarding criteria used did not provide the termination at the same time. However, the algorithm may be stopped after the assigned time for its execution has elapsed, and in such cases the erasure probability is substantially decreased.

Two additional bidirectional algorithms are worth mentioning. Belzile and Haccoun [19] investigated the bidirectional M-algorithm. Since the M-algorithm inherently avoids erasures by its breadth-first nature it still suffers from the correct path loss as its unidirectional version. Another interesting algorithm is the Bidirectional Multiple Stack Algorithm [13]. It additionally decreases the erasure probability of the MSA without compromising the error performance.

A New Bidectional Algorithm

The algorithm [20] uses $2\cdot(L + \mathcal{M} - \mathcal{T})$ processing units (in practice, this number can be somewhat smaller), half of which are intended for forward search and the others for backward (using the reverse code). Processing units, shown in Fig. 2, are arranged in two arrays so that the processing unit at depth l deals only with paths of depths $l - 1$, l, and $l + 1$ (except for the processors at the ends of a row). Each processing unit contains a stack of size M_l (l being the depth of the unit, $l = 1, 2, \ldots, L + \mathcal{M} - \mathcal{T}$) associated with it.

The steps of the algorithm are:

1. Put the root node into the first stack on the F side, and the (unique) terminal node into the first stack on the opposite side, associating them the zero metric.
   The steps 2 – 4 are performed simultaneously in all processing units from both directions.
2. Choose the node with the best metric and eliminate it from the stack. Link it via a tunnel to each of the existing paths in the stack whose length is $L - l + \mathcal{M} - \mathcal{T}$. The total length of the paths $L + \mathcal{M}$ branches. Store the best one into the tentative decision register. If there is already a path in the register, keep the better. Prune the paths remaining in all stacks according to any of discarding criteria used. If all the stacks are emptied in this way, output the tentative decision as the decoder's final decision and terminate the algorithm.
3. Evaluate the metrics of all the successors of the processed path, and eliminate all of them that do not conform to the discarding criteria established. Pass them to the processing unit at the following depth;
4. Receive all the successors from the previous depth and sort them according to their metrics. Return to step 2.

After each tentative decision, several discarding criteria for the paths stored in all processing units are established. The first one is as for the BSA based on the Non-Selection Principle. Since this does not increase the probability of error

significantly, all the paths from any direction that do not satisfy this criterion when compared with the tentative decision should be discarded. Another type of discarding is based on the path distance. Denoting by $d^{\text{inv}}(L - l + \mathcal{M} - \mathcal{T})$ the minimum path metric from any stack from the reverse direction at depths from $L - l + \mathcal{M} - \mathcal{T}$ to $L + \mathcal{M} - \mathcal{T}$, a path of length l and distance d can be discarded whenever $d + d^{\text{inv}}(L - l + \mathcal{M} - \mathcal{T}) \geq d_{\text{TD}}$, where d_{TD} is the total accumulated distance of the tentative decision. This is a maximum-likelihood criterion, and used alone produces the MLD result. Moreover, a path may be discarded whenever it is ranked below the M_l-th place in its current stack (as in the M-algorithm), or when the metric difference between the best path ever from a stack and the path in question is greater than T_l (as in the T-algorithm). Figure 3 shows the immense reduction of the decoding effort in comparison with the BSA.

The algorithm may also be terminated after the time assigned for its execution has elapsed. Since the algorithm in both directions easily follows the correct path till it meets an error burst, and since the distribution of errors in the tunnel does not affect the performance in step 2, no correctable error pattern confined to $\mathcal{T}$ successive branches may affect the number of steps to correct tentative decision (the one obtained by the VA). This would stimulate the choice of longer tunnels if there were not the negative effect of pursuing a great number of unnecessary tunnels when $\mathcal{T}$ approaches $\mathcal{M}$.

Although the space complexity of the new algorithm is linear with the information frame length, there are some practical difficulties with passing on the discarding criteria to all the processing units and analyzing new candidates for the path stored in the tentative decision register. These problems may be dealt easily if transfer of these data is pipelined, too. A certain delay in passing on discarding criteria and candidate tentative decisions thus imposed would yield somewhat worse results.

C. Algorithms That Minimize Symbol Error Rate

a) The BCJR Algorithm

So far, we have considered the algorithms that minimize the error probability of information sequence $\mathbf{i}_{[0,\,L+\mathcal{M})}$. They accomplish it by searching for the "closest" sequence $\mathbf{x}_{[0,\,L+\mathcal{M})}$ according to the metric chosen. However, these algorithms do not necessarily minimize the symbol or bit error rate. It is independently proposed by Bahl *et. al.* and McAdam *et. al.* [10], but more detailed description can be found in [9]. The algorithm is a special case of a more general problem of estimating the a posteriori probabilities of the states and transitions of a Markov source observed through a DMC, i.e. the probabilities

$$P[s_l = i, s_{l+1} = j \mid \mathbf{y}_{[0,L+\mathcal{M})}], \tag{11}$$

or equivalently

$$\sigma_l(i, j) = P[s_l = i, s_{l+1} = j, \mathbf{y}_{[0,L+\mathcal{M})}], \tag{12}$$

where s_l is the state of the trellis during l-th branch. Introducing

$$\begin{aligned} \alpha_l(i) &= P[s_l = i, \mathbf{y}_{[0,l)}], \\ \beta_l(i) &= P[\mathbf{y}_{[l,L+\mathcal{M})} \mid s_l = i], \\ \gamma_l(i, j) &= P[s_{l+1} = j, y_l \mid s_l = i], \end{aligned} \tag{13}$$

it is not hard [9] to show that

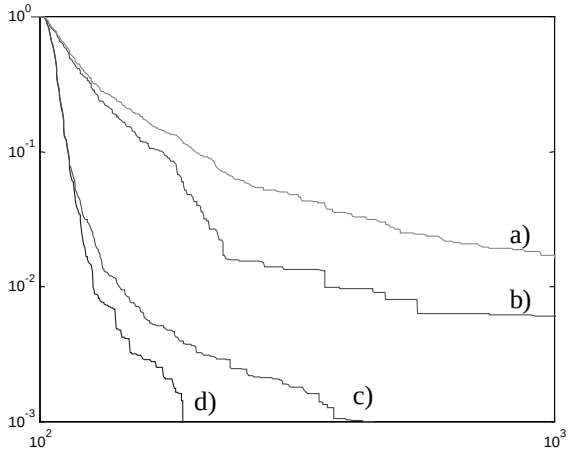

Fig. 3. Computational distributions
a) BSA – in the moment of reaching the final decision
b) BSA – in the moment when algorithm is terminated
c) New alg. – in the moment of reaching the final decision
d) New alg, – in the moment when algorithm is terminated

$$\begin{aligned} \alpha_{l+1}(j) &= \sum_{i=0}^{2^{K\mathcal{M}}-1} \alpha_l(i) \cdot \gamma_l(i, j), \\ \beta_l(j) &= \sum_{i=0}^{2^{K\mathcal{M}}-1} \beta_{l+1}(i) \cdot \gamma_l(j, i), \\ \gamma_l(i, j) &= \sum_{x_l} P[x_l \mid s_l = i, s_{l+1} = j] \cdot \\ &\qquad P[s_{l+1} = j \mid s_l = i] \cdot P[y_l \mid x_l], \\ \sigma_l(i, j) &= \alpha_l(i) \cdot \gamma_l(i, j) \cdot \beta_l(j) \end{aligned} \tag{14}$$

The known initial conditions, i.e. $\alpha_0(i = 0) = 1$, $\alpha_0(i \neq 0) = 0$, $\beta_{L+\mathcal{M}}(i = 0) = 1$, $\beta_{L+\mathcal{M}}(i \neq 0) = 0$, meaning that the initial and terminating state in the trellis is the all zero state, the steps of the algorithm are

1. Initialize $\alpha_0(i)$, and $\beta_{L+\mathcal{M}}(i)$, for i = 0,1,…,$2^{K\mathcal{M}} - 1$ according to (14).
2. As soon as $\mathbf{y}_l$ is received compute $\alpha_l(i)$ and $\gamma_l(i, j)$. Store $\alpha_l(i)$ for all l and i.
3. When the complete sequence $\mathbf{y}_{[0,\,L+\mathcal{M})}$ is received, compute $\beta_l(i)$ using (14), and immediately the probabilities $\sigma_l(i, j)$. Group those $\sigma_l(i, j)$ that have the same information sequence $\mathbf{i}_l$, and choose the largest as the decoder estimate.

The basic problem with the algorithm is that it requires both large storage and great number of computations. All the values of $\alpha_l(i)$ must be stored, which requires almost $(L + \mathcal{M}) \cdot 2^{K\mathcal{M}}$ memory locations. The number of multiplications required for determining the $\alpha_l(i)$ and $\beta_l(i)$ for each l is $2^{K \cdot (\mathcal{M}+1)}$, and there are $2^{K\mathcal{M}}$ additions of 2^K numbers as well. The computation of $\gamma_l(i, j)$ is not costly and can be accomplished by a table lookup. Finally, the computation of all $\sigma_l(i, j)$ requires $2^{K \cdot (\mathcal{M}+1)+1}$ multiplications for each l, and $2^K - 1$ comparisons in choosing the largest $\mathbf{i}_l$, Consequently, this is an algorithm with exponential complexity and in practice can be applied only to decoding trellis codes with short $\mathcal{M}$ and block length L. Nevertheless, it is used for iterative decoding where such requirements can be fulfilled. The main advantage of the algorithm in such cases is decoder's ability to estimate $P[s_{l+1} = j \mid s_l = i]$, which for the possible transi-

timate $P[s_{l+1} = j \mid s_l = i]$, which for the possible transitions equals 2^{-K} only in the first iteration.

a) The SOVA Algorithm

The soft-output Viterbi algorithm (SOVA) [7] is a modification of the Viterbi algorithm designed with the aim to estimate the reliability of every decoded bit by the VA. It is applicable only to punctured codes (of any rate), whose mother codes are of rate 1/N. The VA is used here in its sliding window form, which differs from the ordinary VA in the 3rd step

3. *Path Selection:* From each list at depth *l*, select a path $\mathbf{x}_{[0,l)}$ with the largest metric for the next step, and discard the others. If two or more paths have the same metric, choose the best one at random. Find the best of all the survivor paths, $\mathbf{x}'_{[0,l)}$, and its corresponding information sequence $i'_{[0,l)}$ and release the bit $i'_{l-\delta}$. Go to step 2.

The sliding window VA decodes infinite sequences with delay of δ branches from the last received one. In order to minimize its memory requirements (δ + 1 trellis levels), and achieve bit error rate only insignificantly higher than with finite sequence VA, δ is chosen as $\delta \approx 4\mathcal{M}$.

The reliability (or soft value) of a bit *i*, *L*(*i*), is defined as *L*(*i*) = ln(P[*i* = 0] / P[*i* = 1]). The SOVA further extends the 3rd step in order to obtain this value, in the following way:

3. Path Selection (extension): Let $i^{(j)}_{[0,l-j)}$, $j \in \{0,1,\ldots,\delta-1\}$, be the information sequences which merge with $i'_{[0,l)}$ at depths $l - j$. Their paths have earlier been discarded due to their lower metrics. Let the corresponding metric differences in the merging states be denoted Δ_j, and let $\mathcal{J} = \{j : i^{(j)}_{l-\delta} \neq i'_{l-\delta}\}$. Then $L(i'_{l-\delta}) \approx (1 - 2i'_{l-\delta}) \cdot \min_{j \in \mathcal{J}} \Delta_j$.

Since VA decoding metric can be modified in a way to take into account a priori knowledge of input bit probabilities, the SOVA can be used as soft input-soft output (SISO) block in turbo decoding schemes.

Abstract: An overview of the existing trellis decoding techniques is presented. The paper compares various algorithms that minimize sequence error probability and symbol error probability, focusing on their time-space complexity vs. performance. A new bidirectional algorithm is briefly described.

AN OVERVIEW OF DECODING PROCEDURES FOR TRELLIS CODES – Vojin Šenk, Predrag Radivojac, and Ivan Stanojević.

REFERENCES

[1] S. J. Simmons, "Breadth-First Trellis Decoding with Adaptive Effort," *IEEE Trans. Comm.*, vol. COM-38, No. 1, pp. 3-12, Jan. 1990.

[2] T. Hashimoto, "A List-Type Reduced-Constraint Generalization of the Viterbi Algorithm," *IEEE Trans. Inf. Theory*, vol. IT-33, No. 6, pp. 866-876, Nov. 1987.

[3] J. B. Anderson, S. Mohan, "Sequential Coding Algorithms: A Survey and Cost Analysis," *IEEE Trans. Comm.*, vol. COM-32, No. 2, pp. 169-176, Feb. 1984.

[4] J. L. Massey, Coding and Complexity, CISM courses and lectures No. 216, Springer-Verlag, Wien, 1976.

[5] V. Šenk, "Bistack – A Bidirectional Stack Algorithm for Decoding Trellis Codes," *Proc. of XXXVI Conference on ETAN*, pp. 153-160, Kopaonik, Yugoslavia, 1992.

[6] A. J. Viterbi, J. Omura, Principles of Digital Communication and Coding, McGraw-Hill, Tokyo, 1979.

[7] J. Hagenauer, "Source-Controlled Channel Decoding," . *IEEE Trans. Comm.*, vol. COM-41, pp. 370-380, Feb. 1995.

[8] -, "Applications of Error-Control Coding," *IEEE Trans. Inf. Theory*, vol. IT-44, No. 6, pp. 2531-2560, Oct 1998.

[9] L. R. 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.

[10] P. L. McAdam, L. R. Welch, and C. L. Weber, "M.A.P. bit decoding of convolutional codes," Proc. of ISIT 1972, Asilomar, USA.

[11] R. M. Fano, " A heuristic discussion of probabilistic decoding," *IEEE Trans. Inf. Theory*, vol. IT-9, pp. 64-74, April 1963.

[12] P. R. Chevillat and D. J. Costello Jr., "A Multiple Stack Algorithm for Erasure free Decoding of Convolutional Codes," *IEEE Trans. Comm.*, vol. COM-25, pp. 1460-1470, Dec. 1977.

[13] K. Li, S. Kallel, "A Bidirectional Multiple Stack Algorithm," *IEEE Trans. Comm.*, vol. COM-47, No. 1, pp. 6-9, Jan. 1999.

[14] S. Kallel, K. Li, "Bidirectional Sequential Decoding," *IEEE Trans. Inf. Theory*, vol. IT-43, No. 4, pp. 1319-1326, July 1997.

[15] V. Šenk and P. Radivojac, "The Bidirectional Stack Algorithm," Proc. of ISIT'97, p. 500, Ulm, Germany, July 1997.

[16] V. Šenk and P. Radivojac, "The Bidirectional Stack Algorithm - Simulation Results," Proc. of TELSIKS '95, pp. 349-352, Niš, Yugoslavia, Oct. 1995.

[17] D. Haccoun, M. J. Ferguson, "Generalized Stack Algorithms for Decoding Convolutional Codes," *IEEE Trans. Inf. Theory*, vol. IT-21, No. 6, pp. 638-651, Nov. 1975.

[18] V. Šenk, P. Radivojac, "A Multi-Path Stack Algorithm for Decoding Trellis Codes", Proc. of TELSIKS '97, pp. 780-783, Niš, Yugoslavia, Oct. 1997.

[19] J. Belzile, D. Haccoun, "Bidirectional Breadth-First Algorithms for the Decoding of Convolutional Codes," *IEEE Trans. Comm.*, vol. COM-41, pp. 370-380, Feb. 1993.

[20] V. Šenk, P. Radivojac, "A New Bidirectional Algorithm for Decoding Trellis Codes", submitted for ISIT 2000.

[21] P. Radivojac, V. Šenk, "The Generalized Viterbi-T algorithm," Proc. of XXXIX Conference on ETRAN, Vol. 2, pp. 13-16, Zlatibor, Yugoslavia, June 1995.