KYBERNETIKA — VOLUME *17* (1981), NUMBER 2

HEURISTIC DECODING OF CONVOLUTIONAL CODES

JOSEF KOLÁŘ

The decoding of convolutional codes is presented as a path-searching in corresponding "code graph". Applying some heuristic search techniques new decoding algorithms were obtained, which have improved the decoding process significantly.

1. INTRODUCTION

The Viterbi decoding algorithm for convolutional codes presented in 1967 ([1]) offers a good basis for the conception of a hard-wired decoder designed as a system consisting of a great number of very simple processing units (capable of adding and comparing) with a common memory of sufficient size. The software implementation of this algorithm using a universal monoprocessor system seems to be slow for any practical use even at low transmission rates.

It can be shown that the decoding corresponds to a path finding in a graph and, consequently, the possible application of heuristic search strategies is worth investigating. A systematic "blind" search would examine about $K . 2^L$ different paths in decoding a codeword of length L (K is a code-dependent constant). In the same case, the Viterbi algorithm examines only about $K . L$ paths but the magnitude of K makes this reduction still not sufficient for practical purposes. Heuristic variants of the decoding algorithm given in this paper show a possible way how further reduction can be obtained.

2. CONVOLUTIONAL CODES

A binary convolutional coder of rate $1/n$ can be represented by a shift register of length v coupled with n nonequivalence adders (see Fig. 1). When a given input word $\boldsymbol{u} = u_1 u_2 \dots u_L$ is presented at the input of the shift register, the corresponding codeword $y = y_1 y_2 \dots y_L$ is received at the coder output as follows: for every $k = 1, 2, \dots, L$ the symbol y_k is an n-digit binary block $y_{1k} y_{2k} \dots y_{nk}$. Every bit y_{ik} is received at the output of the i-th adder which computes the nonequivalence sum of certain subset selected from the bits $(u_k, u_{k-1}, \dots, u_{k-n+1})$ stored in the shift

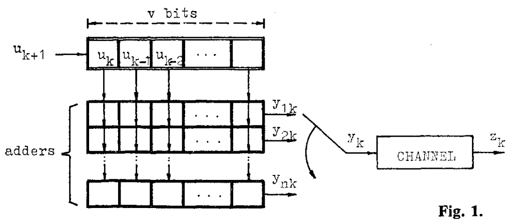
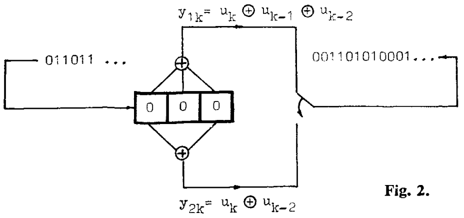


Fig. 1.

register. When the symbol y_k is output, the contents of register is shifted one bit right and the next input bit u_{k+1} enters the register. Fig. 2 shows a convolutional coder for $n = 2$ and $v = 3$.



$y_{1k} = u_k \oplus u_{k-1} \oplus u_{k-2}$
011011 ...
0 0 0
001101010001...
$y_{2k} = u_k \oplus u_{k-2}$

Fig. 2.

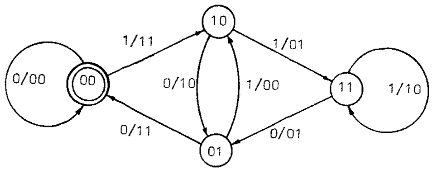


Fig. 3.

The coder can be viewed as finite automaton. The states of this automaton are defined by the contents of the $(v - 1)$ leftmost bits in the shift register. The transition diagram of the coder presented in Fig. 2 is shown in Fig. 3. Every edge of this diagram is labeled by the corresponding pair u_k/y_k. For a given input word the corresponding sequence of transitions describes the coding process. To make this process even more evident we use a time expansion of the transition diagram called the *code graph* (Fig. 4 shows the code graph corresponding to the coder of Fig. 2). We label the edges of the code graph only by the symbols y_k as every upper and lower edge leaving the same node correspond to 0 and 1 input bits, resp. If we denote by $\boldsymbol{x}^k$ the node representing the state x at time k, it is clear that for every input word $\boldsymbol{u}$ there exists a unique path in the code graph which starts in the initial node $\boldsymbol{0}^0$ and ends in some node $\boldsymbol{x}^L$. The corresponding codeword $\boldsymbol{y}$ is received concatenating the labels (output symbols) of individual edges forming this path.

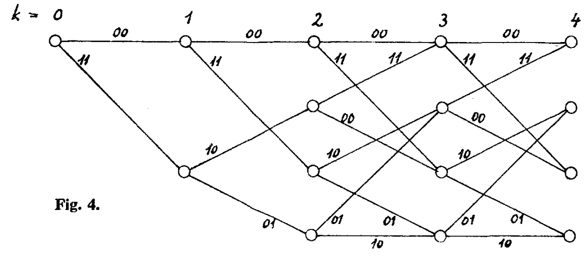


Fig. 4.

In the decoding of a codeword $z = z_1 z_2 \dots z_L$ (received at the output of a transmission channel after some codeword $\boldsymbol{y} = y_1 y_2 \dots y_L$ was presented at its input) a possible noise must be taken into account. In other words, there may not exist any path in the code graph the labeling of which is the codeword z. It can be shown that for a symmetric binary channel the most probable a posteriori estimation of the origin-

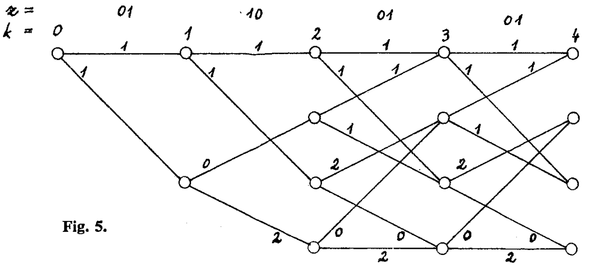


Fig. 5.

al error-free codeword $\boldsymbol{y}$ is defined by such path in the code graph for which the Hamming distance between the words z and $\boldsymbol{y}$ is minimal. To find this path we change the labeling of the code graph so that it defines the Hamming distance between the associated output symbol of each edge and the corresponding received symbol z_k. Fig. 5 shows the code graph labeling for the received codeword $z = 01\ 10\ 01\ 01$.

It follows that the decoding of any codeword z should be transformed to the search of a path between the initial node 0^0 and some node x^L the length of which (in the sense introduced above) is minimal. To eliminate possible ambiguities, every input word presented to the coder is complemented (at the end) by a sequence of $(v - 1)$ zero bits so that all paths converge to the state 0 in the final part of the code graph. We shall include this sequence in the total word length L. Consequently, the decoding problem should be transformed to the search of the shortest path between two explicitly stated nodes of the code graph.

The original decoding algorithm proposed by Viterbi is, in fact, an adaptation of the classical breadth-first search tailored to special properties of the code graph. If we pass by the initial and final parts of the code graph for the simplicity, the central part is created repeating periodically the transition base cell shown in Fig. 6. The

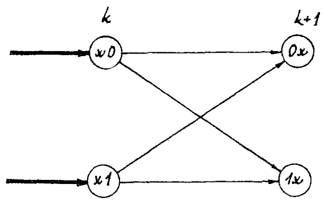


Fig. 6.

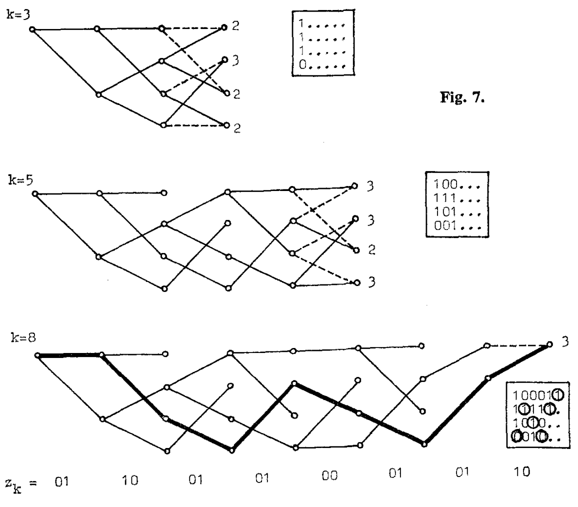


Fig. 7.

Viterbi algorithm iteratively computes the shortest paths (called survivors) to all states at time steps $k = 1, 2, \ldots, L$. Supposing we know all survivors at some time k, the systematic processing of all base cells of the code graph in the following manner will give us all survivors at time $k + 1$:

For every final node $0x$ and $1x$ of the base cell (see Fig. 6) there exist just two possible ways in which the survivors to the initial nodes $x0$ and $x1$ could be prolonged to reach that final node. After the shortest paths to both nodes $0x$ and $1x$ are selected and stored, the processing will pass to the next base cell.

In Fig. 7 we show some steps of the processing when the received codeword is $z = 01\,10\,01\,01\,00\,01\,01\,10$. As every node in the code graph has at most two possible predecessors, the survivors are represented by a binary matrix of $2^{v-1} \,.\, (L - v + 1)$ elements. To make this representation obvious the binary values stored in this matrix and the survivor lengths are also shown in Fig. 7.

3. THE BREADTH-BOUND SEARCH

It can be easily shown that a pure straightforward application of heuristic search strategies would not lead to any computational profit. In spite of an eventual reduction in the number of processed states, it would make the decoding considerably slower due to complicated advance and housekeeping operations. In order to receive better results, we apply some heuristics conserving the most important features of the original algorithm: its simplicity and continuous advance towards the goal.

For every variant of convolutional code and every word length L there exists an upper bound *LIM* of errors that can be corrected by the decoding. We shall make use of this bound to accelerate the decoding process as follows: we change our systematic processing of survivors to non-systematic taking into account only such survivors the length of which does not exceed *LIM*. In other words, we need not to prolong all survivors in full breadth of the code graph but only some limited subset of them. As this subset will vary during the search, it is necessary to create an explicit list of corresponding states in every step and to use this list in the next step. The decoding algorithm based on this idea will be called the *breadth - bound search* (or BBS) in this paper.

Suppose the subset of perspective states is empty after some steps of the search have been accomplished. Consequently, there is no path of length *LIM* or less in the code graph and correct decoding cannot be assured. The common procedure used in communication systems to overcome the problem is to request a re-transmission of the codeword concerned.

The explicit maintaining and processing of limited state subsets complicates slightly the access to the base cells and can be justified only by a significant reduction in the number of processed states. The experiments show that this reduction depends

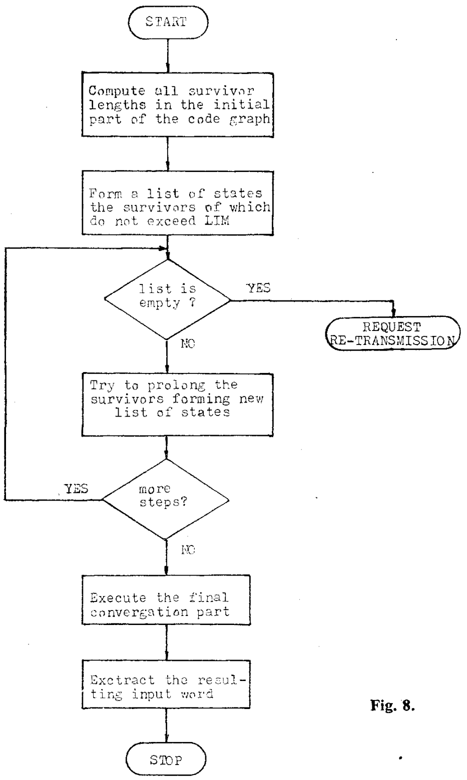

START
Compute all survivor lengths in the initial part of the code graph
Form a list of states the survivors of which do not exceed LIM
list is empty ?
YES
REQUEST RE-TRANSMISSION
NO
Try to prolong the survivors forming new list of states
more steps?
YES
NO
Execute the final convergation part
Exctract the resulting input word
STOP

Fig. 8.

on the number and even more on the position of errors in the received codeword but even in the worst case it overweighs many times the drawback due to the non-systematic processing of base cells. The greater is the number of erroneous bits processed so far, the narrower is the part of the code graph being processed. Consequently, the best total reduction is received when errors are located close to the beginning of the codeword. In our experiments e.g. one bit located at the beginning resulted in greater reduction than 9 errors located at the end.

The implementation of the proposed decoding algorithm represents, naturally, a detailed realization of many parts not explicitly mentioned herein, as e.g. the processing of the initial and final parts of the code graph, efficient Hamming distance calculation, etc. One of the most important features of the algorithm is the splitting of the state subsets into 2 separated parts: the first one receives the states named $0x$ in Fig. 6 (i.e. the states of the interval $\langle 0, 2^{v-2} - 1\rangle$), the second one receives the states named $1x$ (i.e. the states of the interval $\langle 2^{v-2}, 2^{v-1} - 1\rangle$). If the states are processed in the increasing sequence of binary combinations x (otherwise it could not be simply detected whether or not both of the initial states $0x$ and $1x$ of the base cell being processed are included in the subset), the newly created subsets are ordered automatically in the same sense.

Detailed explanation and description of every part of the BBS algorithm would make the paper rather extensive so that only an abbreviated block diagram is presented in Fig. 8.

4. THE BI-DIRECTIONAL HEURISTIC SEARCH

The theory of heuristic search has demonstrated that the search is accelerated in some cases using the bi-directional search strategy. The use of this strategy is possible supposing there is just one goal node explicitly stated and the graph being searched addmits the backward processing. It is easy to show that the path searching in the code graph satisfies both of these conditions.

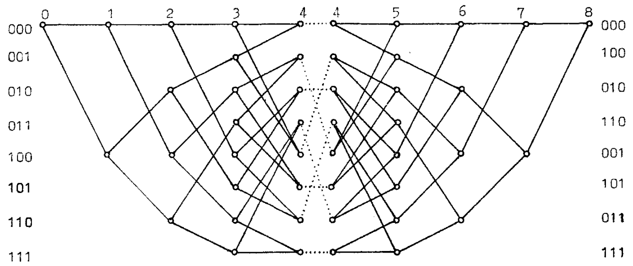


Fig. 9.

In order to make use of the BBS strategy in the backward direction, we have to recorder the states in the second part of the code graph. They will be ordered in the increasing sequence of mirror images of their numbers. In this way, the staes $0x$ and $1x$ used as initial for base celles in the backward direction (see Fig. 6) are assigned consecutive locations – the same is true for the pair $x0$ and $x1$ in the forward direction. Given the values $v = 3$ and $L = 8$, Fig. 9 shows the code graph structure after its second half has been reordered.

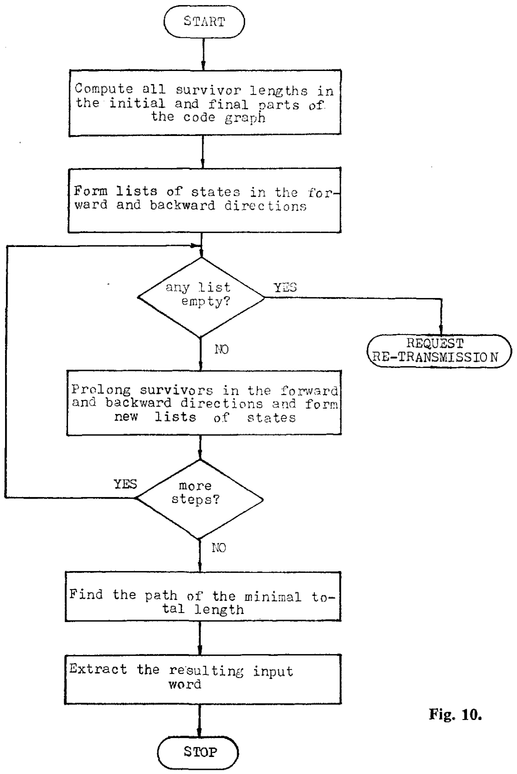

START
Compute all survivor lengths in the initial and final parts of the code graph
Form lists of states in the forward and backward directions
any list empty?
YES
REQUEST RE-TRANSMISSION
NO
Prolong survivors in the forward and backward directions and form new lists of states
more steps?
YES
NO
Find the path of the minimal total length
Extract the resulting input word
STOP

Fig. 10.

Evidently, the code graph obtained this way is perfectly symmetric so that the initial and final expansions can be accomplished, applying the same procedure. In the central part, the steps in the forward and backward directions are formally identical, too. At the moment when both directions meet the path is to be found which minimizes the sum of partial path lengths obtained in the forward and backward directions. It is easy to show that this path will be of the same minimal length as in the uni-directional search.

The criterion used for the search breadth reduction in the uni-directional case can be applied in both directions of the bi-directional search, as well. Even better results will be obtained if we take into account not only the actual path lengths but possible total path lengths. To be able proceed in this way we need some length estimation for the parts that have not been constructed yet. But it is quite easy to receive a simple estimation based on the lengths of paths constructed so far in the opposite direction: the remaining part of every path will not be shorter than the

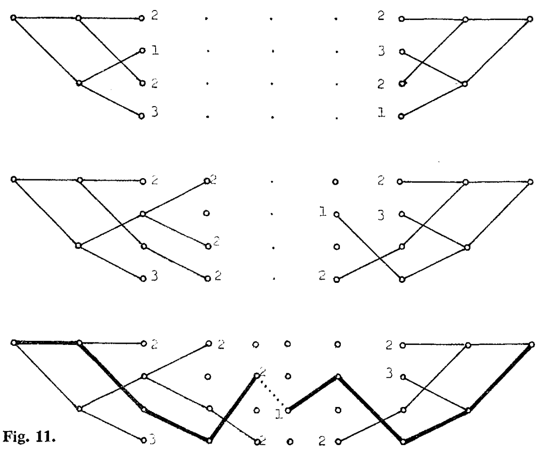


Fig. 11.

minimal partial path length in the opposite direction. If we call this minimal length *MIND* then we shall include in the state subsets only those states which have survivors of length not exceeding *LIM* − *MIND*.

The corresponding algorithm will be called the *bi-directional heuristic search* (or BHS) and its abbreviated block diagram is shown in Fig. 10. The process of decoding of the received codeword z = 01 10 01 01 00 01 01 10 is shown in Fig. 11. As in the BBS case, our description is limited to a brief presentation of the main ideas of the BHS algorithm and more details can be found in [2].

5. EXPERIMENTAL RESULTS

As both of the algorithms presented in this paper are heuristic, it is impossible to express explicitly the number of states (or paths) they proceed during the search. We tried to get some results applying these algorithms to an experimental set of codewords of a convolutional code having the following parameters:

$$n = 3\,, \quad v = 10\,, \quad L = 60\,, \quad LIM = 9$$

$$y_{1k} = u_k \oplus u_{k+3} \oplus u_{k+6} \oplus u_{k+7} \oplus u_{k+8} \oplus u_{k+9}$$

$$y_{2k} = u_k \oplus u_{k+2} \oplus u_{k+3} \oplus u_{k+4} \oplus u_{k+5} \oplus u_{k+7} \oplus u_{k+9}$$

$$y_{3k} = u_k \oplus u_{k+1} \oplus u_{k+2} \oplus u_{k+5} \oplus u_{k+6} \oplus u_{k+8} \oplus u_{k+9}$$

The results of these experiments offer reliable information about the nature of both algorithms. The BBS algorithm is very sensitive to the location of errors in the codeword whereas the BHS algorithm gives almost identical results if errors are located at the end instead of the beginning of the codeword. In case there are no errors at all in the codeword or they are located at the worst position (i.e. at the end for BBS and at the center for BHS), there is practically no significant difference between the two algorithms. Summary results allowing statistical comparison of the Viterbi, BBS and BHS algorithms are given in the following table.

Table

	N_i	N_c	N_f	Total
Viterbi	511	21 504 (100%)	1 022	23 037 (100%)
BBS	511	1 354 (6·3%)	1 022	2 887 (12·5%)
BHS	511	957 (4.45%)	511	1 979 (8·59%)

(Numbers of nodes processed in the initial — N_i, central — N_c and final — N_f part of the code graph represent mean values for the set of 40 examples.)

From these results we see that the BBS algorithm processed only about 12·5% of all states processed by the Viterbi algorithm. The BHS algorithm is even better processing about 8·6% of states. Both algorithms introduced here have a common feature which seems rather paradoxical: the more errors are in the codeword (within the acceptable limit) the more effective is the decoding. Consequently, the codewords received without any error at all are decoded the slowest of all. Further effort is to be devoted to possible elimination of this paradox.

ACKNOWLEDGMENTS

The research reported in this paper was partially carried out when the author was at C.I.E.A. del I.P.N. (National Politechnical Institute), Mexico City, Mexico. The author would like to acknowledge the support supplied by the CONACYT and C.I.E.A. during his stay.

(Received May 7, 1980.)

REFERENCES

[1] A. J. Viterbi: Error Bounds for convolutional codes and an asymptotically optimal decoding algorithm. IEEE Trans. on Inform. Theory *IT-13* (1967), 260—269.

[2] J. Kolář: Heuristický přístup k dekódování konvolučních kódů. Habilitation thesis (in Czech), FEL ČVUT, Prague 1979.

RNDr. Josef Kolář, CSc., katedra počítačů, fakulta elektrotechnická ČVUT (Department of Computers, Faculty of Electrical Engineering — Czech Technical University), Karlovo nám. 13, 131 35 Praha 2. Czechoslovakia.