The Golay-Viterbi Concatenation Scheme

E. R. Berlekamp[1]
Communications Systems Research Section

This article examines in detail the Golay-Viterbi concatenation scheme which has been proposed for use for the nontelevision scientific data portion of the Mariner Jupiter/Saturn telemetry. The simplest form of the scheme makes no use of the memory in the noise caused by the Viterbi decoder; in this article we will demonstrate that it is possible to utilize this memory to obtain improved performance.

It has been proposed that scientific data on the MJS'77 project be given more noise protection than the TV data and that this can be achieved by using the Golay code, interleaved to a depth of j ($j = 10$ to 25) to achieve an overall block length of $24j$. The $24j$ bits in any overall block may be conveniently arranged in j rows of 24 bits each, where each row is a code word in the Golay code. The bits are transmitted consecutively, column by column. The outer channel through which these blocks are transmitted contains a convolutional encoder and a Viterbi decoder. Even if the space channel over which the inner convolutional code is transmitted is memoryless, the phenomenon of error propagation in the Viterbi decoder will cause the outer channel to be bursty. However, the interleaved nature of the outer code distributes an error burst among the j different Golay codes, so that most bursts do not cause more than a few errors in any particular Golay code word. It is therefore possible to correct many common error patterns by decoding the outer code row by row.

Of course, there are some error patterns which this procedure will fail to correct. The most common such error patterns will cause one or more rows to be received as a word with highly improbable syndrome, corresponding to an error pattern of weight 4 or more with no simple burst explanation. When this rare event occurs, we suggest a refined algorithm for redecoding the entire block of $24j$ bits. This algorithm would probably be programmed on a computer rather than implemented in hardware. Since it would be used only on those rare blocks on which an error was detected by the Golay code, the refined algorithm need not be as fast as the conventional row-by-row decoding algorithm operating on the output of the Viterbi decoding hardware.

[1]Consultant to JPL from U. C. Berkeley.

The most effective attack is to go back to the original data and reexamine the decisions made by the Viterbi decoder in more detail. The goal of this reexamination is to erase those bits on which the decoding decisions were close. Once this has been done, it will be quite easy for the Golay code to correct the erasures as well as the small number of errors which might remain. It is well known that if there are no errors present, any linear binary code with minimum distance d can correct *all* patterns of $d - 1$ or fewer erasures, and hence, that the extended (24, 12) Golay code can correct any pattern of not more than 7 erasures. The easiest way to accomplish this erasure decoding up to minimum distance is to set all erasure bits to 0, try to decode, then set all erased bits to 1, and try to decode again. Of course, one attempt or the other must succeed in correcting an "error" pattern of weight less than $d/2$ in the erased positions.

It is less well known that linear codes typically have much greater erasure correction capabilities than indicated by the minimum distance. In particular, when there are no errors, the Golay code is not only capable of correcting *all* patterns of 7 erasures, it can also correct 322/323 of the patterns of 8 erasures as well as most patterns of 9 or 10 erasures, 286/323 of the possible patterns of 11 erasures and over half of the patterns of 12 erasures. In other words, erasures are *less* than half as difficult to correct as errors. A program for correcting the erasures need only solve simultaneous linear equations over GF(2).

We now comment on the problem of simulating a refined Viterbi decoder which will erase those message bits about which it is not too sure. Recall that the Viterbi decoder makes two kinds of errors, the so-called "decoding" errors, which arise when its estimate of the last bit in the encoder's shift register is incorrect, and "memory truncation errors," which arise because its memory of 4 or 5 constraint lengths is occasionally insufficient. The latter type of error appears to be considerably less frequent than the former, and if necessary could be further reduced by using a longer memory in the simulator.

To detect the "decoding" errors, one might design a modified Viterbi decoder in which each original register was replaced by a list of L registers holding the L most probable message sequences consistent with the appropriate hypothesis about the current contents of the encoder's shift register. Even with $L = 2$, this complicates the simulating program considerably, because *most* decisions made by the Viterbi decoder are very close. However, this is usually not because the decoder is on the verge of making an error, but because the decision is between two very improbable sequences somewhere in the trellis, and neither of these sequences is destined to reach the output anyway.

Since the original sequence will have already been processed by the Linkabit decoder anyway, the simplest way to simulate a decoder with list 2 would be to begin by re-encoding the sequence that was decoded on the first pass and subtract it from the original.

The simulator's task of finding good second-choice message sequences is now simplified by its a priori knowledge that the first choice will be the all-zero sequence. Because of this fact, the simulator need not differ from the conventional Viterbi decoder except in the number of its outputs. In addition to the estimated message sequence, which we know will be all-zero except in the rare events when the simulator is able to correct a memory truncation error, there are three other outputs which are of considerable importance. These are the likelihood of the message register containing the (first choice) all-zero sequence, the likelihood of the message register containing the (second choice) 1000 . . . 0 sequence, and the full previous message sequence corresponding to this second choice. This sequence represents the most probable error event under the hypothesis that the error event ends with the bit about to pass out of the message register.

When the likelihoods of the two candidates are close, then a likely error event has been detected and the appropriate bits of the decoded message sequence should be erased.

Of course, those error events which occur when two or more wrong paths are more probable than the correct one will not be erased by the algorithm just described. Such events would cause a decoding error even if the decoder were allowed a list of two candidates. When the rate of the code is sufficiently low (or, equivalently, if the signal-to-noise ratio is sufficiently high), then a list-of-two error of this type is much less probable than a conventional list-of-one error. However, when the rate of the code is sufficiently high (or, equivalently, the signal-to-noise ratio is sufficiently low), then most list-of-one errors are also list-of-two errors.

Fortunately, the three outputs of the simulator contain sufficient information to determine, with high probability, which type of error has occurred. When the likelihoods of both candidates are relatively high, then a list-of-two

error is very unlikely and a decoding error can almost surely be prevented by erasing the second-choice candidate message pattern. For example, if this is . . . 00010101, then only the three bits indicated by the ones need be erased. The two zero bits within the likely error event were decoded correctly, independently of whether this likely error event actually occurred. However, if the likelihoods of both candidates are relatively low, then the selected choice represents the best of a bad lot, and a list-of-two error is almost as likely as a list-of-one error. Very probably, the third-, fourth-, and fifth-place choices rank very close to the second. Under these circumstances, the only way the Viterbi simulator can be fairly sure of making no error is to erase *all* of the bits within the burst corresponding to the second choice, and possibly a few previous bits as well. I cannot think of any good simple algorithm to determine how far back to extend the erasure burst. One sophisticated approach would be to couple this simulator with another simulator which decoded the convolutional code *backwards in time*. Just as the outputs of the forward-in-time Viterbi simulator give good estimates of the probabilities of error events ending at each message digit, the outputs of the reverse-in-time Viterbi simulator would give good estimates of the probabilities of error events beginning at each message digit. By combining the two, a good estimate of the location and extent of list-of-two error events could be obtained. The message bits within such a span could then be erased.

There remains the problem of selecting the values of likelihoods which should cause the simulator to erase. The selection of these parameters should be governed by the erasure and error-correct capabilities of the outer code.

With 3 erasures in a block of length 24, the Golay code can still correct any pattern of two or fewer errors. A single error can be corrected in the presence of any pattern of 5 erasures and most patterns of 6 erasures. Hence, if the Viterbi simulator erases up to 15 to 25% of the bits in a long block, there is still an excellent chance that the outer interleaved Golay code will be able to correct these erasures and any error events that were undetected by the simulator as well. If the outer code still fails to correct the erased Viterbi output, then there are still further, even more complicated attacks that might be tried. For example, one might try to use the constraint that all bits in a candidate Viterbi error event corresponding to a likely second choice must be the same. However, the number of blocks not decoded correctly by the refined outer decoder when suitable bits of its input have been erased is so small that such further refinements may not be necessary.

Other work on causing the Viterbi decoder to output erasures has been done by Clark and Davis (Ref. 1).

Reference

1. Clark, G. C., Jr., and Davis, R. C., "Two Recent Applications of Error-Correction Coding to Communications Systems Design," *IEEE Trans. on Comm. Tech.*, Vol. 19, pp. 856–863, 1971.