Maximum likelihood decoding of block codes on their tail-biting trellises

Priti Shankar K Sasidharan P N A Kumar B S Rajan

IISc-CSA-TR-2003-3

http://archive.csa.iisc.ernet.in/TR/2003/3/

Computer Science and Automation
Indian Institute of Science, India

February 2003

Maximum likelihood decoding of block codes on their tail-biting trellises

Priti Shankar* K.Sasidharan* P.N.A. Kumar*

B.S.Rajan†

*Department of Computer Science and Automation, Indian Institute of Science, Bangalore,

†Department of Electrical Communication Engineering, Indian Institute of Science, Bangalore,

priti@csa.iisc.ernet.in, bsrajan@ece.iisc.ernet.in

February 5, 2003

Abstract

An exact maximum likelihood decoding algorithm is presented for tail-biting trellises for block codes. The algorithm works in two phases. The first phase is a Viterbi trial on the tail-biting trellis, which provides estimates that the second phase which is an A* algorithm uses to compute the most likely codeword. Detailed simulation results are presented for two block codes. These indicate that the estimates computed in the first phase greatly reduce the search space in the second phase.

1 Introduction

It has been known for a long time that block codes have compact combinatorial descriptions in the form of tail-biting trellises [14, 8]. A tail-biting trellis can have considerably fewer states than a conventional trellis for many well known block codes [1]. For example a conventional trellis for the extended (24,12) Golay code has 256 states at its mid-point whereas there exists a twelve section tail-biting trellis which has only 16 states at each level[1]. Given this fact, it is worthwhile investigating decoding algorithms on such trellises. The usual approaches to the maximum likelihood decoding of codes on tail-biting trellises involve several Viterbi trials for a few cycles around the loop until a winning codeword is detected. Maximum-likelihood decoding algorithms on tail-biting trellises were proposed by Ma and Wolf[8], Solomon and van Tilborg[14], Zigangirov and Chepyzhov[17], Wang and Bhargava[15], Cox and Sundberg[2] and by Shao et al.[12]. Most of these are approximate decoding algorithms. In this correspondence we describe an exact maximum likelihood two phase decoding algorithm and study its performance on the (24,12) extended binary Golay code and the (31,21) binary BCH code. Section 2 presents some background; section 3 presents the decoding algorithm. Finally section 4 presents the results of simulations and concludes the correspondence.

2 Background

It is well known[14, 3, 13, 11] that linear or group tail-biting trellises can be viewed as the superposition of subtrellises that share states at certain time indices, the start state considered to have time index 0 and any other state having time index i, if the paths from the start state to it have i edges. The structure of a tail-biting trellis can be described as follows. Let there be l *start*(equivalently *final*) states.

Each start state corresponds to a subcode, which consists of all codewords that begin and end at that state. Thus there are a total of l subcodes. These subcodes all have identical subtrellises. It is therefore logical to call all these subtrellises *copies* of one another, as they are structurally identical, and differ only in edge labels. The subtrellis containing the all 0 codeword defines a subcode which is a group, and the remaining subtrellises correspond to cosets of the group. The l subtrellises share states at various time indices. An obvious possibility that exists for exact decoding on a tail-biting trellis, is to perform Viterbi decoding on each of the l subtrellises and then choose the codeword with minimal cost among the l survivors at time index n in each subtrellis. (From now on, we call these *winning* codewords). Previous work on ML decoding for tail-biting trellises has concentrated on approximate algorithms. We briefly describe some of them now. One of the first algorithms reported was Bar-David's probabilistic algorithm, the basic steps of which are described below:

1. Choose an arbitrary start state s_i.
2. Decode using the maximum likelihood decoder which finds the best path from that starting state to all ending states.
3. Check if the winning path is a codeword. If so, stop, otherwise else go to the next step.
4. Use the previous ending state as the new starting state. Check if this starting state has been tried before. If it has, go to step 1 else go to step 2.

The algorithm outputs the best winning path if the number of trials reaches the number of subtrellises.

An alternate suboptimum decoding algorithm has been proposed in [8] is the two-step algorithm. The first step in this algorithm is to obtain an ordered list of the starting states using an algebraic method called "continued fractions". The second step is to perform Viterbi runs using each entry in the list as the starting state. Both these algorithms require on the average, a number of trials which is half the number of starting states, which makes them impractical unless this number is small.

Wang and Bhargava [15] have observed that a Viterbi trial on the tail-biting trellis often gives the most likely codeword, the probability of this event rising rapidly with the SNR. However if this is not the case, their algorithm goes through successive phases, each phase removes from consideration the best candidate codeword survivor found so far(if one is found) and checks if the current winner is better than the previous winner. If it is not the decoding stops. If it is, the process is repeated. If the result of a phase does not give a winning codeword then the number of start states is reduced by approximately half by deleting in order, start states that begin the maximum number of winning noncodewords, until the number of states drops to $\lceil N/2 \rceil$ or $\lfloor N/2 \rfloor$ where N is the current number of states, and also the corresponding final states of that subtrellis. The result is the maximum likelihood path. Wang and Bhargava have carried out simulations to show that the number of Viterbi trials is considerably fewer than that for the first two algorithms at high signal to noise ratios. However the complexity of their algorithm gets close to the brute force complexity of decoding separately on all subtrellises and finding an overall winner for noisy channels.

Several authors have conceived of the idea of tail-biting decoding by going round and round the tail-biting trellis[14, 7, 17, 6]. These algorithms transfer information from trial to trial and start the decoding at all states in parallel. Cox and Sundberg[2] present an algorithm which they term a Circular Viterbi Algorithm(CVA). Here the received block of coded symbols with soft decision information is used, and the Viterbi algorithm is used for continuous decoding of a sequence of repeated blocks, thus simulating the situation where the information is transmitted repeatedly in block form. The decoder can start in all states with the same metric start value. All available soft information of a decoding trial is transferred to the next trial. This is why the algorithm is called circular, since the end of one block ties into the repeated version of the same block. The decoder either stops after a fixed number of rounds or or if two successive decoding trials give the same block. The authors have reported results very close to the exact ML results for a much lower decoding complexity.

Shao et. al[12] describe two algorithms, the Wrap-Around Viterbi(WA-V) algorithm and the Iterative Bidirectional Viterbi Decoding(IBVD). The WA-V makes consecutive "rounds"of the trellis until a stopping condition is reached. At each round at each state two metrics are computed, the accumulative state metric(ASM) and the State Metric Gain(SMG). The ASM at a state in the i^{th} iteration is the total path metric of a survivor that originates at a start node at the beginning of the first iteration and terminates at that state in the i^{th} iteration. The state metric gain of a state in the i^{th} iteration measures the path metric of the survivor at that state in the i^{th} iteration beginning from the start state of the path in the i^{th} iteration. At each iteration when the final states are reached, the path with the best metric is chosen as the winning path. If the winning path corresponds to a codeword, then decoding stops, else it is stored and the next iteration is begun. Decoding stops when the first winning codeword path is found or after a preset maximum number of iterations. The IBVD algorithm processes a TBT from opposite directions with two decoders called the left and the right decoder. Both decoders execute the WAV algorithm and they collaborate to make a decoding decision. At each state the SMG's of the left and right decoders at that state are added to give the composite SMG, which is simply the path metric of the survivor of length equal to the number of sections of the TBT passing through that state. The path thus obtained is called a composite path. If the composite path at the end of the last section corresponds to a codeword decoding stops, else find the best composite path and store it in memory. The iteration continues till a codeword path is found or until a preset maximum number of iterations is reached.

The algorithm we propose here is an exact algorithm which uses the first Viterbi trial on the TBT to gather estimates that are used in the second phase. Under certain conditions the second phase is not needed at all. Simulations show that the estimates gathered during the first phase are very effective in reducing the search space during the second phase and the computational overheads are not excessive even for noisy channels.

3 The New Decoding Algorithm

We first define the notation we will be using. Let T_B be the tail-biting trellis and T_C the conventional trellis for the code under consideration. Let $s_1, s_2, \ldots s_l$ be the set of start states and $f_1, f_2, \ldots f_l$ be the set of final nodes of the tail-biting trellis. The l sub-trellises are denoted by $T_1, T_2, \ldots T_l$, with sub-trellis T_j corresponding to sub-code C_j. The estimate or lower bound for a shortest path from a node u in a sub-trellis to the final node f_j in that sub-trellis is denoted by $h(u, f_j)$. A Viterbi trial on the tail-biting trellis finds shortest paths to the destination nodes $f_1, f_2, \ldots f_l$ from *any* of the source nodes $s_1, s_2, \ldots s_l$. A *survivor* at any intermediate node u is a shortest path from a source node s_i to u. A *winning* path or a *winner* at node f_j is a survivor at node f_j and its cost is denoted by $h(s_j, f_j)$, as it turns out to be a lower bound on the cost of a shortest path from s_j to f_j. (Note that such a path need not start at s_j.) We recall that the tail-biting trellis may be viewed as a superposition of l subtrellises where l is the number of start(or final) states. Let us term the codeword that would be the final survivor if Viterbi decoding was performed on subtrellis T_j alone, as the T_j*-codeword survivor*. We term a survivor on the tail-biting trellis, which corresponds to an $s_i - f_i$ path, a *codeword survivor*, and one that corresponds to a $s_i - f_j$ path, $i \neq j$, a *non-codeword survivor*. Two observations about survivors may be made at this point.

Observation 1: A non-codeword survivor at f_j on the tail-biting trellis, always has cost which is a lower bound on the T_j-codeword survivor.

Observation 2: If there is a non-codeword survivor at f_j on the tail-biting trellis, that has cost greater than that of some codeword survivor on the tail-biting trellis, then the T_j-codeword survivor cannot be the winning codeword.

Observation 1 follows from the fact that the survivor at f_j is a non-codeword, because at some intermediate node, the prefix of the path corresponding to the T_j-codeword survivor was 'killed' by a prefix of a path originating at some other start node. This latter prefix had a lower cost. Hence the cost of the non-codeword survivor is actually less than that of the actual T_j-codeword survivor.

Observation 2 is a direct consequence of observation 1, which implies that a T_j-codeword survivor has a cost that is at least as great as that of a non-codeword survivor at f_j. Thus if decoding on the tail-biting trellis yields a codeword survivor at node f_k with lower cost than the non-codeword survivor at f_j, the T_j-codeword survivor cannot be the winning codeword.

This brings us to phase 2 of the decoding algorithm. This has to be executed if and only if there is no codeword survivor in the tail-biting trellis, which is of lowest cost among all survivors at final nodes. The subtrellises that participate in phase 2 are therefore only those which have non-codeword survivors, which have a cost less than any codeword survivor in the tail-biting trellis. Let us call these *residual trellises.*

At any given point of execution of phase 2, the algorithm is travelling on exactly one of the residual subtrellises, called the *current* subtrellis. This is the one that looks most promising in its potential to deliver the minimum cost path. The execution at all othe residual subtrellises is kept in a suspended state, the best among them to be taken up at some point, if the current subtrellis is no longer found to have the best potential at that point. The *potential* of a node of a subtrellis is defined as a sum of two quantities. If u is the node currently being examined in the current subtrellis, say T_j, the potential of u decides whether the algorithm continues on this subtrellis or takes up another one; this is the sum of the cost of the shortest path from s_j to u and an *estimate* of the shortest path from u to f_j. The first phase provides us with these estimates for all the *start* nodes and these estimates are computed for each node as we move forward node by node in a subtrellis. The first path to reach a final node in the second phase, corresponds to the winning codeword. (Note that it is bound to be a codeword, as the algorithm will be working on a single subtrellis, namely the current subtrellis at that point). We now describe the algorithm more formally. We first describe the A^* algorithm.

3.1 The A^* Algorithm

The A^* algorithm is well known in the literature on artificial intelligence [4] and that it can be used for decoding was demonstrated in [5]. The A^* algorithm uses, in addition to the path length from the source to the node u, an estimate $h(u, f)$ of the shortest path length from the node u to the goal node in guiding the search. Let $L_T(u, f)$ be the shortest path length from u to f in T. Let $h(u, f)$ be any lower bound such that $h(u, f) \leq L_T(u, f)$, and such that $h(u, f)$ satisfies the following inequality, i.e, for u a predecessor of v, $l(u, v) + h(v, f) \geq h(u, f)$. If both the above conditions are satisfied, then the algorithm A^*, on termination, is guaranteed to output a shortest path from s to f. The algorithm is given below.

Algorithm A^*
Input : A trellis $T = (V, E, l)$ where V is the set of vertices, E is the set of edges and $l(u, v) \geq 0$ for edge (u, v) in E, a source vertex s and a destination vertex f, and an estimate $h(u, f)$ for the shortest path from u to f for each vertex $u \in V$.
Output : The shortest path from s to f.
/* $p(u)$ is the cost of the current shortest path from s to u and $P(u)$ is a current shortest path from s to u */

begin
    $S \leftarrow \emptyset, \quad \bar{S} \leftarrow \{s\}, \quad p(s) \leftarrow 0, \quad P(u) \leftarrow (), \forall u \in V, \quad p(u) = +\infty, \forall u \neq s;$
    repeat
        Let u be the vertex in $\bar{S}$ with minimum value of $p(u) + h(u, f)$.
        $S \leftarrow S \cup \{u\}; \quad \bar{S} \leftarrow \bar{S} \setminus \{u\};$
        if $u = f$ *then return* $P(f)$;
        for each $(u, v) \in E$ *do*
            if $v \notin S$ *then*
                begin
                    $p(v) \leftarrow \min(p(u) + l(u, v), previous(p(v)));$
                    if $p(v) \neq previous(p(v))$ *then* append (u, v) to $P(u)$ to give $P(v)$;
                    $(\bar{S}) \leftarrow (\bar{S}) \cup \{v\};$
                end
    forever
end

3.2 The Two Phase Algorithm

The two phases of the algorithm are described below. (All assertions in italics are proved in the next subsection).

Phase 1.
Execute a Viterbi decoding algorithm on the tail-biting trellis, and obtain survivors at each node. As we saw earlier,each survivor at a node u has a cost which is a lower bound on the cost of the least cost path from s_j to u in an $s_j - f_j$ path passing through u, $1 \leq j \leq N$. *If there exists a value of k for which an $s_k - f_k$ path is an overall winner then this is the shortest path in the original trellis T_C.* If this happens decoding is complete. If no such $s_k - f_k$ path exists go to Phase 2.

Phase 2

1. Consider only subtrellises T_j such that the winning path at T_j is an $s_i - f_j$ path with $i \neq j$ (i.e at some intermediate node a prefix of the $s_j - f_j$ path was "knocked out" by a shorter path originating at s_i), and such that there is no $s_k - f_k$ path with smaller cost. Let us call such trellises *residual trellises.* Let the estimate $h(s_j, f_j)$ associated with the node s_j be the cost of the survivor at f_j obtained in the first phase.

2. Create a heap of r elements where r is the number of residual trellises, with current estimate $h(s_j, f_j)$ with minimum value as the top element. Let j be the index of the subtrellis with the

minimum value of the estimate. Remove the minimum element corresponding to T_j from the heap and run the A^* algorithm on trellis T_j (called the *current trellis*). For a node u, take $h(u, f_j)$ to be $h(s_j, f_j) - cost(survivor(u))$ where $cost(survivor(u))$ is the cost of the survivor obtained in the first phase. The quantity $h(u, f_j)$ *satisfies the two properties required of the estimator in the* A^* *algorithm.*

3. At each step, compare $p(u) + h(u, f_j)$ in the current subtrellis with the top value in the heap. If at any step the former exceeds the latter (associated with subtrellis, say, T_k), then make T_k the current subtrellis. Insert the current value of $p(u) + h(u, f_j)$ in the heap (after deleting the minimum element) and run the A^* algorithm on T_k either from start node s_k (if T_k was not visited earlier) or from the node which it last expanded in T_k. Stop when the goal vertex is reached in the current subtrellis.

In the best case (if the algorithm needs to execute Phase 2 at all) the search will be restricted to a single residual subtrellis; the worst case will involve searching through all residual subtrellises.

3.3 Proofs of Assertions in the Decoding Algorithm.

Lemma 3.1 *If there exists a value of k for which an $s_k - f_k$ path is an overall winner in the overlayed trellis T_v, then this is the shortest path from source to goal in the original trellis T_C.*

Proof: The set of paths in T_B from time index 0 to time index n is a superset of paths from the source to the goal vertex in T_C, as only $s_k - f_k$ paths in T_B correspond to paths in T_C. Thus if an $s_k - f_k$ path is an overall winner in T_B, it must be a winner in T_C as well. □

Lemma 3.2 *The quantity $h(u, f_j)$ defined in the algorithm satisfies the following two properties :*

1. $h(u, f_j) \leq L_{T_j}(u, f_j)$
2. $l(u, v) + h(v, f_j) \geq h(u, f_j)$ *where* (u, v) *is an edge.*

Proof:

1. $h(u, f_j) = cost(survivor(f_j)) - cost(survivor(u))$
   Also $cost(survivor(f_j)) \leq cost(survivor(u)) + L_{T_j}(u, f_j)$, from which the result follows.

2. To prove: $l(u, v) + h(v, f_j) \geq h(u, f_j)$

$$\begin{aligned} \text{LHS} &= l(u, v) + h(v, f_j) \\ &= l(u, v) + h(s_j, f_j) - cost(survivor(v)) \end{aligned}$$

   If survivor at v is survivor at u concatenated with edge (u, v), then

$$\begin{aligned} \text{LHS} &= l(u, v) + h(s_j, f_j) - cost(survivor(u)) - l(u, v) \\ &= h(u, f_j) \end{aligned}$$

   On the other hand if survivor at v is not a continuation of the survivor at u,

$$cost(survivor(v)) \; < \; cost(survivor(u)) \; + \; l(u,v)$$
$$cost(survivor(v)) \; - \; l(u,v) \; < \; cost(survivor(u))$$
$$\text{or,} \quad h(s_j, f_j) \; - \; cost(survivor(v)) \; + \; l(u,v) \; > \; h(s_j, f_j) \; - \; cost(survivor(u))$$
$$\text{or,} \quad h(v, f_j) \; + \; l(u,v) \; > \; h(u, f_j)$$
$$\text{Therefore,} \quad l(u,v) \; + \; h(v, f_j) \; \geq \; h(u, f_j)$$

□

4 Simulation results

Since the performance of this algorithm in terms of the total number of operations performed during decoding is critically dependent on the quality of estimates obtained in the first phase, we performed simulations on tail-biting trellises to evaluate these estimates for the extended (24,12) binary Golay code (for which a minimal tail-biting trellis is displayed in [1] and the binary BCH (31,21)code for which a minimal tail-biting trellis is constructed using techniques described in [11]. The channel was modeled as an AWGN channel.

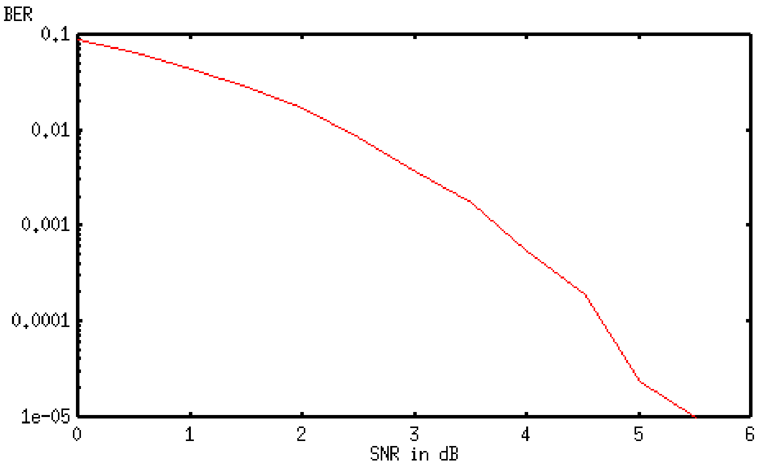

Figure 1: Bit error rate for the Golay code

The state complexity profile of a trellis is the vector $(s_0, s_1, \ldots s_{n-1})$ where s_i is the logarithm to the base q (the alphabet size) of the number of nodes at time index $i, 1 \leq i < n$ The trellis parameters for the two codes are given below.

1. Extended (24,12) binary Golay code.
Tail Biting Trellis:
Number of Branches = 384, Number of States = 192
State Complexity Profile = (4,4,4,4,4,4,4,4,4,4,4,4)

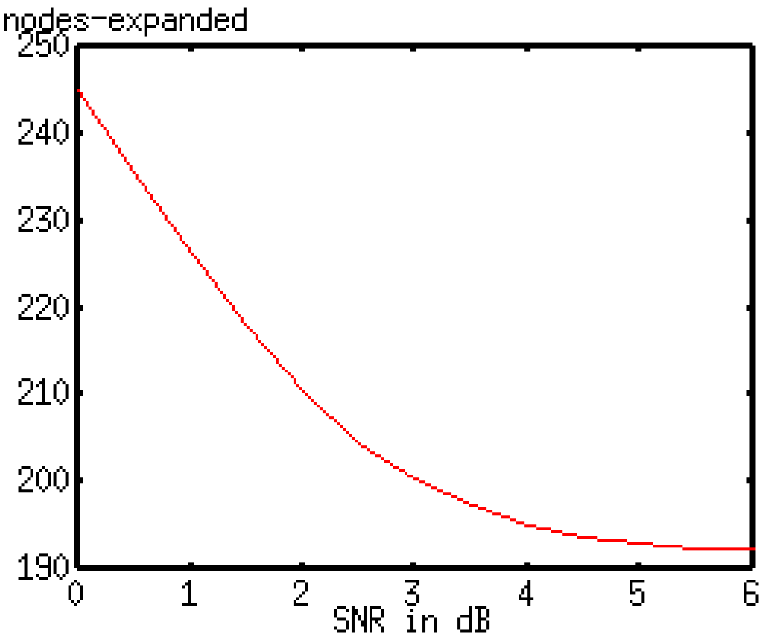

Figure 2: Number of Viterbi updates for the Golay code

Conventional 12-section Trellis:
Number of Branches = 2728, Number of States = 1065
State Complexity Profile = (0,2,4,6,6,8,8,8,6,6,4,2).

2. **BCH(31,21) code over GF(2)**
Tail Biting Trellis:
Number of Branches = 5760, Number of States = 3520
State Complexity Profile = (6,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,6,6,6,6,6,6)
Conventional Trellis:
Number of Branches = 26620, Number of States = 14334
State Complexity Profile = (0,1,2,3,4,5,6,7,8,9,10,10,10,10,10,10,10,10,10,10,10,10,9,8,7,6,5,4,3,2,1)

The following quantities were measured as a function of the signal to noise ratio in dB.

1. The bit error rate.

2. The total number of Viterbi Updates(VU) in the first and second phases. By a Viterbi Update of a node is meant an add, compare and select operation at that node.(The notation is from [2]). For the Viterbi algorithm on the conventional trellis, the number of VU's is the number of nodes in the trellis. For our algorithm, the number of VU's in the first pass is the number of nodes of the tail-biting trellis, and in the second pass, the number of VU's is not fixed. For the Golay code even at 0 dB we see that the total number of VU's in the first and second passes is only 25% more than those in the Viterbi trial on the tail-biting trellis in the first pass, which indicates that the first phase provides excellent estimates. The same is true of the BCH code, for which the

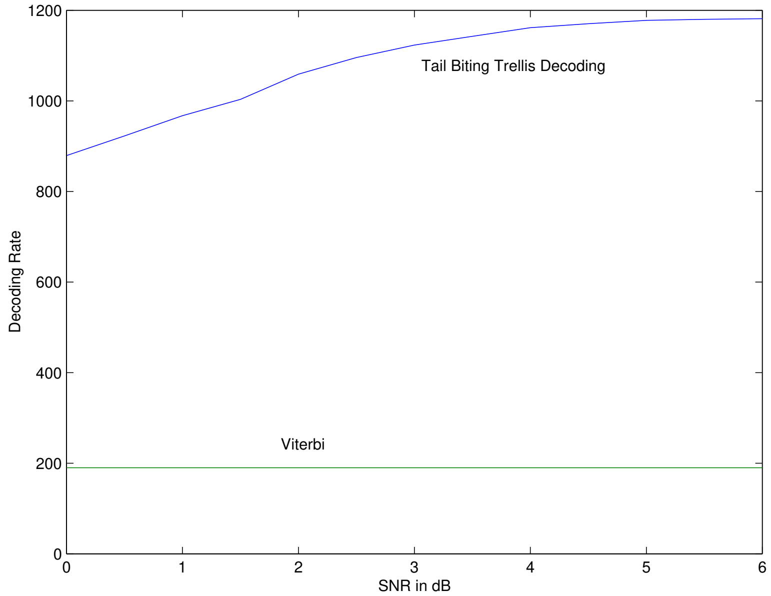

Figure 3: Rates of decoding using the Viterbi algorithm and the two phase algorithm for the Golay code

corresponding figure at 0dB is 30%.

3. The decoding rates on both the conventional trellis as well as the tail-biting one. The number of operations when performing Viterbi decoding on a conventional trellis with E edges and V vertices is $2E - V$[9] added to the cost of initializations. The asymptotic decoding complexity using our algorithm on a tail-biting trellis is $2E' - V' + S_0^2 + S_0 \times n$ added to the cost of initializations where E' and V' are the number of edges and vertices in the tail-biting trellis, S_0 is the number of start states in the tail-biting trellis and n is the length of the block code. The last two terms in the expression for the tail-biting trellis are the complexities of sorting the survivors at the final nodes and tracing back the paths to the start states in order to identify residual trellises. A second pass is rarely needed at high signal to noise ratios.

4. The number of residual trellises. It turns out that the average number of resudual trellises for the Golay code after the first pass is 6 at an SNR of 0(i.e. roughly 40% of the total number of sub-trellises) and that for the BCH code is around 50% of the total number of subtrellises. However, this number drops rapidly with increasing SNR and for the Golay code is less than 1 at 4 dB.

5. The probability of a second pass. The probability of a second pass is seen to drop steeply with the SNR. This confirms the observations of [15] who note that for high values of SNR the average number of Viterbi trials is less than 2.

6. The number of switches between trellises given that there is a second pass. What this measures is the number of times our algorithm moves between subtrellises in the search for the one that contains the final decoded codeword. Here again the number of switches between trellises drops rapidly with increasing SNR.

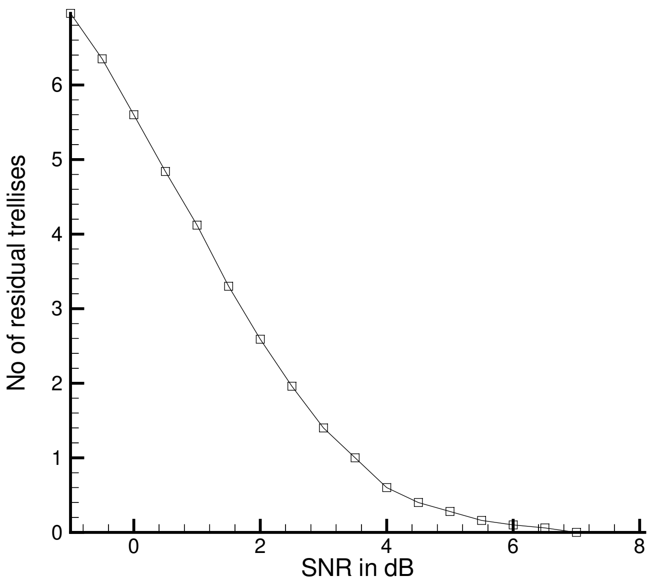

Figure 4: Average number of residual sub-trellises in the second pass for the Golay code

The results of simulation support the claim that the computational overheads of the exact algorithm are not excessive even for noisy channels. However, the space overheads are high. In addition to the space required for storing the estimates of the first phase, we also need an array of size S_0 at each node to store the exact values of the path metric computed at each visited node during the second phase. Further each node has to store a bit vector of length S_0 to indicate which subtrellises it is a member of.

Finally, it would be interesting to see a modification of this algorithm for MAP decoding.

References

[1] A.R.Calderbank, G.David Forney,Jr., and Alexander Vardy, Minimal Tail-Biting Trellises: The Golay Code and More, *IEEE Trans. Inform. Theory* **45**(5) July 1999, pp 1435-1455.

[2] R.V. Cox and C.V. Sundberg, An Efficient Adaptive Circular viterbi Algorithm for Decoding Generalized Tailbiting Convolutional Codes, *IEEE Transactions on Vehicular Technology* **43**(1), February 1994, pp 57-68.

[3] K.Deshmukh, P.Shankar, A. Dasgupta and B.Sundar Rajan, On the many faces of block codes, in *Proceedings of STACS 2000, LNCS 1770* , (Lille, France, February 2000), pp 53-64.

[4] P.E. Hart, N.J. Nilsson, and B. Raphael, A formal basis for the heuristic determination of minimum cost paths, *IEEE Trans. Solid-State Circuits* **SSC-4**, 1968, pp 100-107.

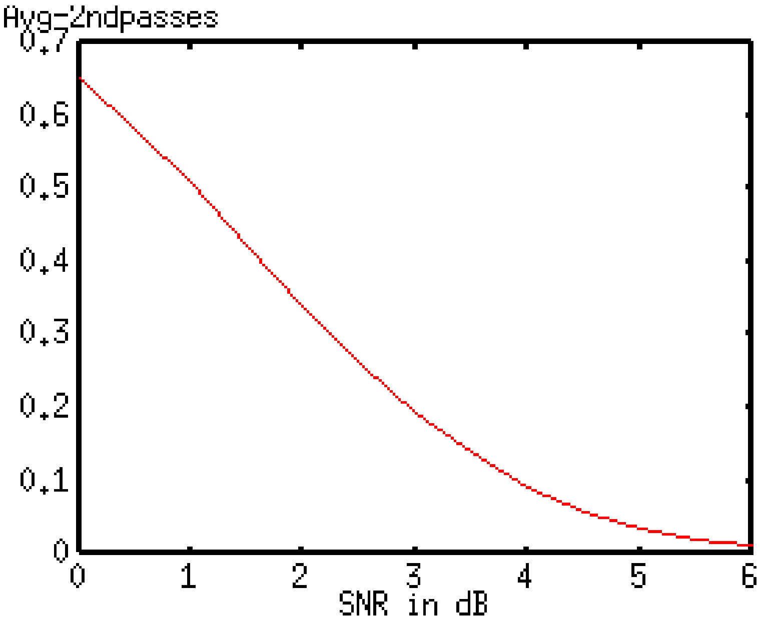

Figure 5: Probability of a second pass for the Golay code

[5] Y.S.Han, C.R.P.Hartmann, and C.C.Chen, Efficient Priority-First Search Maximum-Likelihood Soft-Decision Decoding of Linear Block Codes, *IEEE Trans. Inform.Theory* **39**(5),Sept. 1993, pp 714-729.

[6] B.D. Kudrashyov, Decoding of block codes obtained from convolutional codes, *Problemy Peredachi Informatsii*,**26**(2), pp 18-26, April-June 1990(in Russian). English Translation, Plenum Publishing Corporation, Oct 1990.

[7] R.S. LeFever, A new class of powerful, viterbi decodable block codes, presented at the 1988 Tactical Communications conference, Fort Wayne, Indiana, 1988 Conf. Rec., pp 304-307.

[8] J.H.Ma and J.K.Wolf, On tail-biting convolutional codes, *IEEE Trans. Commun.*,**34**,February 1986, pp 104-111.

[9] R.J. McEliece, On the BCJR Trellis for Linear Block Codes. *IEEE Trans. Inform. theory* **42**(4), July 1996, pp 1072-1092.

[10] P. Shankar, P.N.A. Kumar, Harmeet Singh and B.S. Rajan, Minimum Tail-Biting Trellises for Certain Cyclic Block Codes are Easy to Construct, in *Proceedings of the* 28^{th} *International Symposium on Automata, Languages and Programming(ICALP 01), LNCS 2086*, Crete, Greece.

[11] Yaron Shany and Yair Be'ery, Linear Tail-Biting Trellises, the Square-Root Bound, and Applications for Reed-Muller Codes, *IEEE Trans. Inform. Theory,* **46**(4), July 2000, pp 1514-1523.

[12] R.Y.Shao, Shu Lin and M.P.C. Fossorier, Decoding of Codes Based on Their Tail Biting Trellises, in *Proc. 2000 Int. Symp. Inform. Theory*, Sorrento, Italy, pp 342.

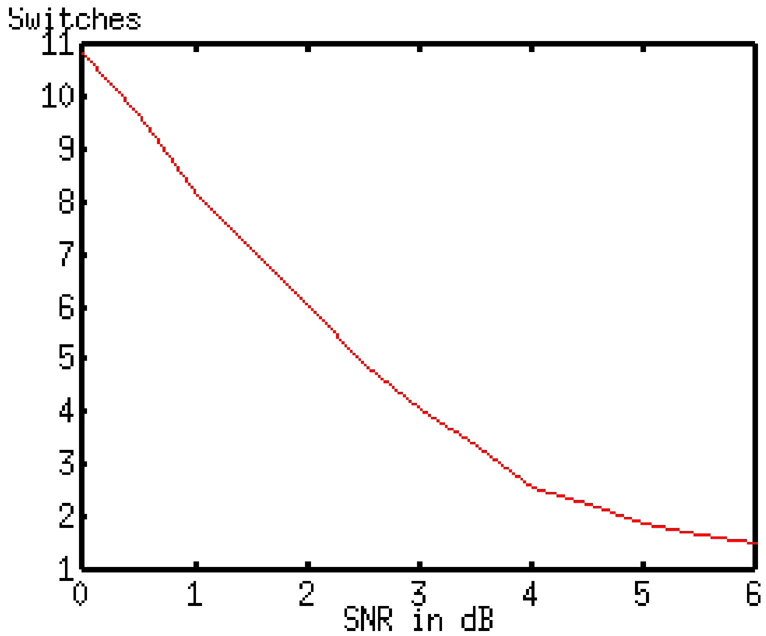

Figure 6: Average number of switches between sub-trellises for the Golay code

[13] Shu Lin and R.Y.Shao, General Structure and construction of Tail Biting Trellises for Linear Block Codes, in *Proc. 2000 Int. Symp. Inform. Theory*, Sorrento, Italy, pp 117.

[14] G.Solomon and H.C.A. van Tilborg, A connection between block and convolutional codes, *SIAM J. Appl. Math.*, **37**, October 1979, pp 358-369.

[15] Q.Wang and V.K.Bhargava, An efficient maximum-likelihood decoding algorithm for generalized tail-biting codes including quasi-cyclic codes, *IEEE Trans. Commun.*, **37**, August 1989, pp 875-879.

[16] N.Wiberg, H.-A. Loeliger and R.Kotter, Codes and iterative decoding on general graphs, *Euro. Trans. Telecommun.,* **6** pp513-526, Sept 1995.

[17] K.S.Zigangirov and V.V. Chepyzhov, Study of decoding tail-biting convolutional codes, in *Proc. Swedish-Soviet Workshop on Information Theory*(Gotland, Sweden, Aug. 1989), pp 52-55.

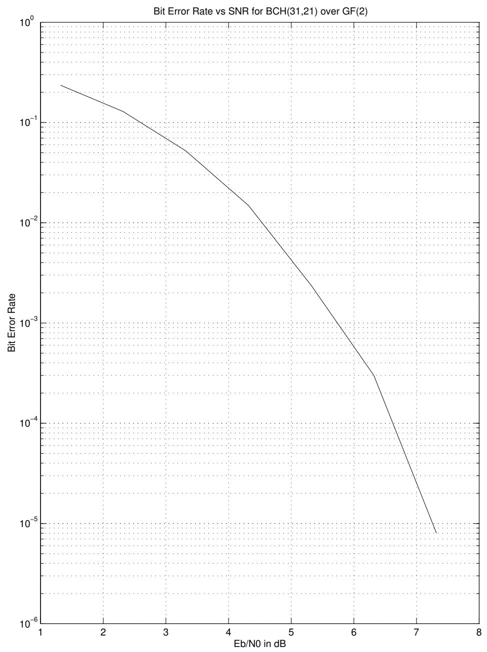

Figure 7: Bit error rate for the binary (31,21) BCH code

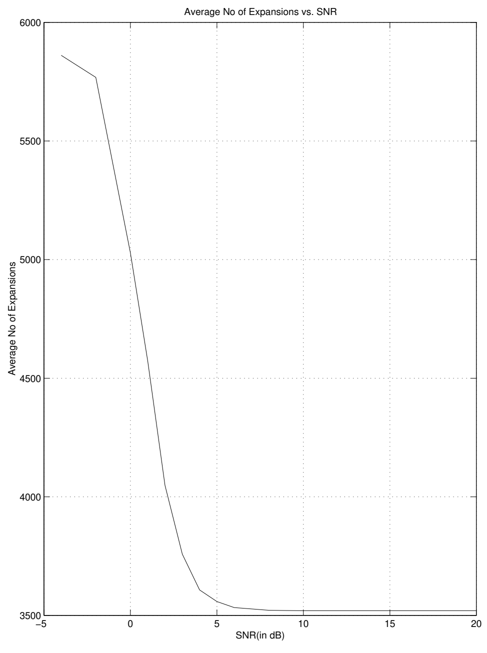

Figure 8: Average number of Viterbi updates for the binary (31,21) BCH code

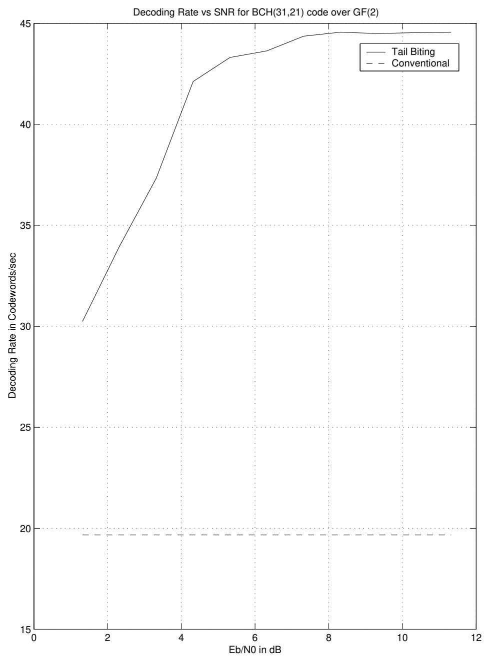

Figure 9: Rates of decoding using the Viterbi algorithm and the two phase algorithm for the binary (31,21) BCH code

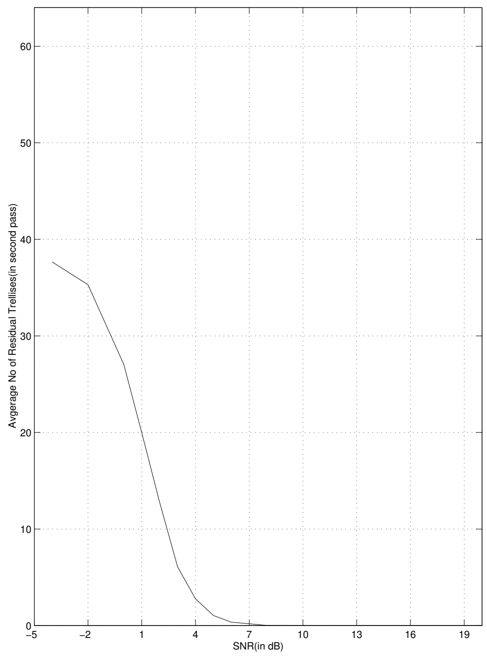

Figure 10: Average number of residual trellises for the binary (31,21) BCH code

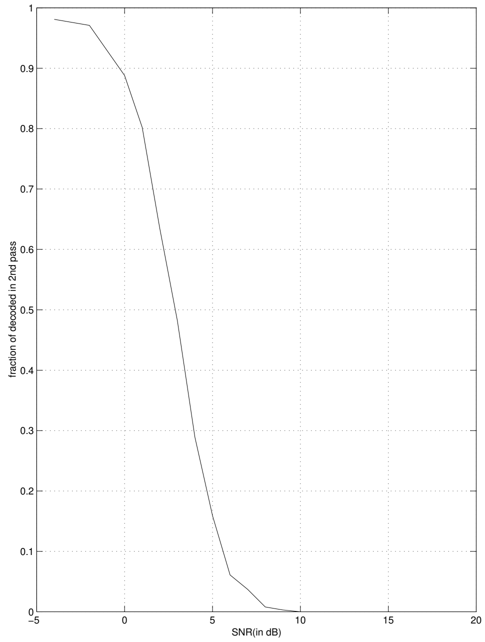

Figure 11: Probability of a second pass for the binary (31,21) BCH code

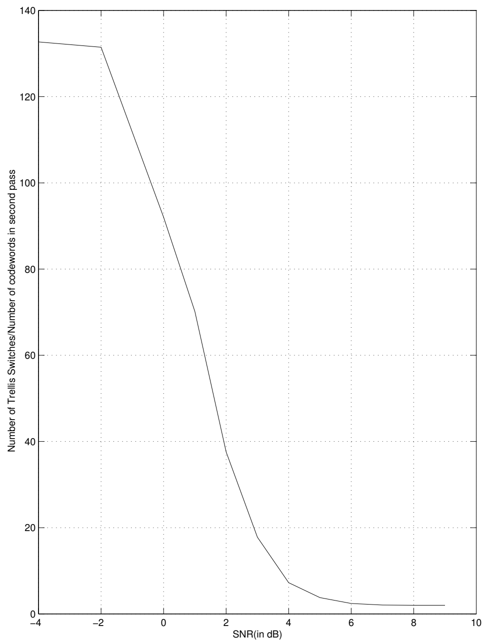

Figure 12: Average number of switches between sub-trellises for the binary (31,21)BCH code