A Package for the Implementation of Block Codes as Finite Automata

Priti Shankar[1], K.Sasidharan[1], Vikas Aggarwal[1], and B.Sundar Rajan[2]

[1] Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560012, India
[2] Department of Electrical Communication Engineering, Indian Institute of Science, Bangalore 560012, India
{priti,sasi,vikas}@csa.iisc.ernet.in, bsrajan@ece.iisc.ernet.in

Abstract. We have implemented a package that transforms concise algebraic descriptions of linear block codes into finite automata representations, and also generates decoders from such representations. The transformation takes a description of the code in the form of a $k \times n$ generator matrix over a field with q elements, representing a finite language containing q^k strings, and constructs a minimal automaton for the language from it, employing a well known algorithm. Next, from a decomposition of the minimal automaton into subautomata, it generates an overlayed automaton, and an efficient decoder for the code using a new algorithm. A simulator for the decoder on an additive white Gaussian noise channel is also generated. This simulator can be used to run test cases for specific codes for which an overlayed automaton is available. Experiments on the well known Golay code indicate that the new decoding algorithm is considerably more efficient than the traditional Viterbi algorithm run on the original automaton.

Keywords: block codes, minimal trellis, decoder complexity, subtrellis overlaying.

1 Introduction

The theory of finite state automata has many interesting connections to the field of error correcting codes [13]. After the early work on trellis representations of block codes[1,22,15,18,7], there has recently been a spate of research on minimal trellis representations of block error correcting codes[8,12,17]. Trellis descriptions are *combinatorial* descriptions, as opposed to the traditional *algebraic* descriptions of block codes. A minimal trellis for a linear block code is just the transition graph for the minimal finite state automaton which accepts the language consisting of the set of all codewords. With such a description, the decoding problem reduces to finding a cheapest accepting path in such an automaton(where transitions are assigned costs based on a channel model). However, trellises for many useful block codes are often too large to be of practical value. Of immense interest therefore, are *tail-biting* trellises for block codes, recently introduced in [3],

S. Yu and A. Păun (Eds.): CIAA 2000, LNCS 2088, pp. 279–291, 2001.

which have reduced state complexity. The strings accepted by a finite state machine represented by a trellis are all of the same length, that is the *block length* of the code. Coding theorists therefore attach to all states that can be reached by strings of the same length l, a *time index* l. Conventional trellises use a linear time index, whereas tail-biting trellises use a circular time index. It has been observed[21] that the maximum state complexity of a tailbiting trellis at any time index can drop to the square root of the maximum state complexity of a conventional trellis for the code, thus increasing the potential practical applications of trellis representations for block codes. It is shown in [6] that tailbiting trellises can be viewed as *overlayed automata*, i.e. a superimposition of several identically structured finite state automata, so that states at certain time indices are shared by two or more automata. This view leads to a new decoding algorithm that is seen to be more efficient than the traditional Viterbi algorithm. Some preliminary details[10] are available for the construction of minimal tailbiting trellises from conventional trellises. The full theory is currently being worked out[11].

In this paper, we describe a software package that provides the following facilities.

1. It constructs a minimal finite state automaton(conventional trellis) for a linear block code from a concise algebraic description in terms of a generator matrix using the algorithm of Kschischang-Sorokine[12]. This involves two steps: First, the conversion of the generator matrix into *trellis oriented* form. This is a sequence of operations similar to Gaussian elimination. Second, the generation of the minimal trellis as the *product trellis* of smaller trellises. Note that the algebraic description is concise, consisting of a $k \times n$ matrix with entries from a field with q elements. The language consists of q^k strings.
2. Given a set of identically structured automata(subtrellises), that can be superimposed to produce an overlayed automaton(tailbiting trellis), it produces a decoder for the block code using a new algorithm described in [6].
3. Given the parameters of an additive white Gaussian noise(AWGN) channel it simulates the decoder on the overlayed automaton and outputs the decoded vector and other statistics for the range of signal to noise ratios(SNR) requested by the user.

Simulations on the hexacode[5], and on the practically important Golay code, the tailbiting trellises of which are both available[3], indicate that there is a significant gain in decoding rate using the new algorithm on the tailbiting trellis over the Viterbi algorithm on the conventional trellis.

We hope to augment the package with a module to convert from a minimal conventional trellis to a minimal tailbiting trellis when the technique(stated to be polynomial time in [10]) becomes available.

Section 2 describes conventional trellises for block codes and the Kschischang-Sorokine algorithm used to build the minimal trellis; section 3 defines tailbiting trellises and overlayed automata; section 4 describes the decoding algorithm; section 5 describes our implementation and presents some results obtained by running our decoder on a tailbiting trellis for the Golay code. Finally section 6 concludes the paper.

2 Minimal Trellises for Block Codes

In block coding, an *information sequence* of symbols over a finite alphabet is divided into *message blocks* of fixed length; each message block consists of k information symbols. If q is the size of the finite alphabet, there are a total of q^k distinct messages. Each message is encoded into a distinct *codeword* of n $(n > k)$ symbols. There are thus q^k codewords each of length n and this set forms a *block code* of length n. A block code is typically used to correct errors that occur in transmission over a communication channel. A subclass of block codes, the *linear block codes* has been used extensively for error correction. Traditionally such codes have been described *algebraically*, their algebraic properties playing a key role in *hard decision* decoding algorithms. In hard decision algorithms, the signals received at the output of the channel are *quantized* into one of the q possible transmitted values, and decoding is performed on a block of symbols of length n representing the received codeword, possibly corrupted by some errors. By contrast, *soft decision* decoding algorithms do not require quantization before decoding and are known to provide significant coding gains [4] when compared with hard decision decoding algorithms. That block codes have efficient *combinatorial* descriptions in the form of *trellises* was discovered in 1974 [1]. Other early seminal work in the area appears in [22] [15] [7] [18]. For background on the algebraic theory of block codes, readers are referred to the classic texts [14, 2,16]; for trellis structure of block codes, [19] is an excellent reference.
Let F_q be the field with q elements. It is customary to define linear codes algebraically as follows:

Definition 1. *A* linear block code C *of length* n *over a field* F_q *is a* k*-dimensional subspace of an* n*-dimensional vector space over the field* F_q *(such a code is called an* (n, k) *code).*

The most common algebraic representation of a linear block code is the generator matrix G. A $k \times n$ matrix G where the rows of G are linearly independent and which generate the subspace corresponding to C is called a *generator matrix* for C. Figure 1 shows a generator matrix for a $(4, 2)$ linear code over F_2. A

$$\mathbf{G} = \begin{bmatrix} 0\,1\,1\,0 \\ 1\,0\,0\,1 \end{bmatrix}$$

Fig. 1. Generator matrix for a $(4, 2)$ linear binary code

general block code also has a *combinatorial* description in the form of a *trellis*. We borrow from Kschischang and Sorokine [12] the definition of a trellis for a block code.

Definition 2. *A trellis for a block code C of length n, is an edge labeled directed graph with a distinguished* root *vertex s, having in-degree* 0 *and a distinguished* goal *vertex f having out-degree* 0*, with the following properties:*

1. *All vertices can be reached from the root.*
2. *The goal can be reached from all vertices.*
3. *The number of edges traversed in passing from the root to the goal along any path is n.*
4. *The set of n-tuples obtained by "reading off" the edge labels encountered in traversing all paths from the root to the goal is C.*

The length of a path (in edges) from the root to any vertex is unique and is sometimes called the *time index* of the vertex. One measure of the size of a trellis is the total number of vertices in the trellis. It is well known that minimal trellises for linear block codes are unique [18] and constructable from a generator matrix for the code [12]. Such trellises are known to be *biproper*. Biproperness is the terminology used by coding theorists to specify that the finite state automaton whose transition graph is the trellis, is deterministic, and so is the automaton obtained by reversing all the edges in the trellis. (Formal language theorists call such languages bideterministic languages). In contrast, minimal trellises for non-linear codes are, in general, neither unique, nor deterministic [12]. Figure 2 shows a trellis for the linear code in figure 1.

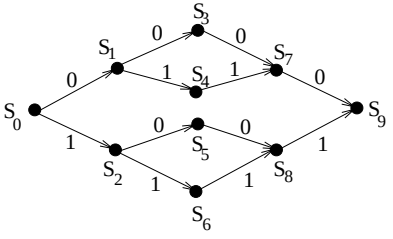

Fig. 2. A trellis for the linear block code of figure 1 with $S_0 = s$ and $S_9 = f$

2.1 Constructing Minimal Trellises for Block Codes

We briefly describe the algorithm given in [12] for constructing a minimal trellis,implemented in our package. An important component of the algorithm is the trellis product construction, whereby a trellis for a "sum" code can be obtained as a *product* of component trellises. Let T_1 and T_2 be the component trellises. We wish to construct the trellis product $T_1.T_2$. The set of vertices of the product trellis at each time index, is just the Cartesian product of the vertices of the component trellis. Thus if i is a time index, $V_i(T_1.T_2) = V_i(T_1) \times V_i(T_2)$. Consider $E_i(T_1) \times E_i(T_2)$, and interpret an element

$((v_1, \alpha_1, v_1'), (v_2, \alpha_2, v_2'))$ in this product, where v_i, v_i' are vertices and α_1, α_2 edge labels, as the edge $((v_1, v_2), \alpha_1 + \alpha_2, (v_1', v_2'))$ where + denotes addition in the field. If we define the i^{th} section as the set of edges connecting the vertices at time index i to those at time index $i+1$, then the edge count in the i^{th} section is the product of the edge counts in the i^{th} section of the individual trellises.

Before the product is constructed we put the matrix in *trellis oriented form* described now. Given a non zero codeword $C = (c_1, c_2, \ldots c_n)$, $start(C)$ is the smallest integer i such that c_i is non zero. Also $end(C)$ is the largest integer for which c_i is nonzero. The *span* of C is $[start(C), end(C)]$. By convention the span of the all 0 codeword $\mathbf{0}$ is []. The minimal trellis for the binary $(n, 1)$ code generated by a nonzero codeword with span $[a, b]$ is constructed as follows. There is only one path up to $a-1$ from index 0, and from b to n. From $a-1$ there are 2 outgoing branches diverging(corresponding to the 2 multiples of the codeword), and from $b-1$ to b, there are 2 branches converging. For a code over F_q there will be q outgoing branches and q converging branches. It is easy to see that this is the minimal trellis for the 1-dimensional code.
To generate the minimal trellis for C we first put the trellis into *trellis oriented form*, where for every pair of rows, with spans $[a_1, b_1], [a_2, b_2], a_1 \neq b_1$ and $a_2 \neq b_2$. We then construct individual trellises for the k 1-dimensional codes as described above, and then form the trellis product. Conversion of a generator matrix into trellis oriented form requires a sequence of operations similar to Gaussian elimination, applied twice. In the first phase, we apply the method to ensure that each row in the matrix starts its first nonzero entry at a time index one higher than the previous row. In the second phase we ensure that no two rows have their last nonzero entry at the same time index. We see that the generator matrix displayed earlier is already in trellis oriented form. The complexity of the Kschischang-Sorokine algorithm is $O(k.n + s)$ for an (n, k) linear code whose minimal trellis has s states.

3 Tailbiting Trellises

We borrow the definition of a tailbiting trellis from [10].

Definition 3. *A tailbiting trellis $T = (V, E, F_q)$ of depth n is an edge labelled directed graph with the following property. The vertex set can be partitioned as follows: $V = V_0 \cup V_1 \cup \ldots V_{n-1}$, such that every edge in T either begins at a vertex of V_i and ends at a vertex of V_{i+1} for some $i = 1, 2, \ldots n-2$ or begins at a vertex of V_{n-1} and ends at a vertex of V_0.*

The notion of a minimal tailbiting trellis is more complicated than that of a conventional trellis. The ordered sequence, $(|V_0|, |V_1|, \ldots |V_{n-1}|)$ is called the *state complexity profile* of the tailbiting trellis. For a given linear code C, the state complexity profiles of all tailbiting trellises for C form a partially ordered set under componentwise comparison. A trellis T is said to be smaller or equal to another trellis T', denoted by $T \leq_S T'$ if $|V_i| \leq |V_i'|$ for all $i = 0, 1, \ldots n-1$. It is

smaller if equality does not hold for all i in the expression above. We say that a tailbiting trellis is minimal under $\leq_S$ if a smaller trellis does not exist. For conventional trellises the minimal trellis is unique. However, for tailbiting trellises, there are, in general, several nonisomorphic minimal trellises that are incomparable with one another. Yet another ordering on tailbiting trellises is given by the *product ordering* $\leq_P$. This is a total ordering. $T <_P T'$ iff $\prod_{i=0}^{n-1} |V_i| < \prod_{i=0}^{n-1} |V_i'|$. It is stated in [10] that $T <_S T' \iff T <_P T'$. An outline for constructing a minimal tailbiting trellis for a linear block code is given in [10]. The complexity is stated to be $O(n^2)$ for a code of length n. The detailed theory in under preparation[11]. Figure 4 is a tailbiting trellis for the linear code of figure 1. Let $S_{max}(T)$ denote the maximum number of states of trellis T at any time index, when the index is allowed to range from 0 to n.
It is shown in [6] that a tailbiting trellis can be viewed as an *overlayed automaton*. This is a somewhat more natural view, we believe, and it also leads to an efficient decoding algorithm on tailbiting trellises.

3.1 Tailbiting Trellises as Overlayed Automata

An overlayed trellis has been introduced in [6], and we give the definition below. Let C be a linear code over a finite alphabet. Let $C_0, C_1, \ldots C_l$ be a partition of the code C, such that C_0 is a subgroup of C under the operation of componentwise addition over the structure that defines the alphabet set of the code(usually a field or a ring), and $C_1, \ldots C_l$ are cosets of C_0 in C. Let $C_i = C_0 + h_i$ where $h_i, 1 \leq h_i \leq l$ are coset leaders, and let C_i have minimal trellis T_i. The subcode C_0 is chosen so that the maximum state complexity is N (occurring at some time index, say, m), where N divides M the maximum state complexity of the conventional trellis at that time index. The subcodes $C_0, C_1, \ldots C_l$ are all disjoint subcodes whose union is C. Further, the minimal trellises for $C_0, C_1, \ldots C_l$ are all structurally identical and two way proper. (That they are structurally identical can be verified by relabeling a path labeled $g_1 g_2 \ldots g_n$ in C_0 with $g_1 + h_{i_1}, g_2 + h_{i_2} \ldots g_n + h_{i_n}$ in the trellis corresponding to $C_0 + h_i$ where $h_i = h_{i_1} h_{i_2} \ldots h_{i_n}$.) We therefore refer to $T_1, T_2, \ldots T_l$ as *copies* of T_0.

Definition 4. *An* overlayed *proper trellis is said to exist for C with respect to the partition $C_0, C_1, \ldots C_l$ where $C_i, 0 \leq i \leq l$ are subcodes as defined above, corresponding to minimal trellises $T_0, T_1, \ldots T_l$ respectively,with $S_{max}(T_0) = N$, iff it is possible to construct a proper trellis T_v satisfying the following properties:*

1. *The trellis T_v has $l + 1$ start states labeled $[s_0, \emptyset, \emptyset, \ldots \emptyset], [\emptyset, s_1, \emptyset \ldots \emptyset] \ldots [\emptyset, \emptyset, \ldots \emptyset, s_l]$ where s_i is the start state for subtrellis $T_i, 1 \leq i \leq l$.*
2. *The trellis T_v has $l + 1$ final states labeled $[f_0, \emptyset, \emptyset, \ldots \emptyset], [\emptyset, f_1, \emptyset, \ldots \emptyset], \ldots [\emptyset, \emptyset, \ldots \emptyset, f_l]$, where f_i is the final state for subtrellis T_i, $0 \leq i \leq l$.*
3. *Each state of T_v has a label of the form $[p_0, p_1, \ldots p_l]$ where p_i is either $\emptyset$ or a state of $T_i, 0 \leq i \leq l$. Each state of T_i appears in exactly one state of T_v.*

4. *There is a transition on symbol a from state labeled* $[p_0, p_1, \ldots p_l]$ *to* $[q_0, q_1, \ldots q_l]$ *in* T_v *if and only if there is a transition from* p_i *to* q_i *on symbol a in* T_i*, provided neither* p_i *nor* q_i *is* $\emptyset$*, for at least one value of i in the set* $\{0, 1, 2, \ldots l\}$.
5. *The maximum width of the trellis* T_v *at an arbitrary time index* $i, 1 \leq i \leq n-1$ *is at most* N.
6. *The set of paths from* $[\emptyset, \emptyset, \ldots s_j, \ldots \emptyset]$ *to* $[\emptyset, \emptyset, \ldots, f_j, \ldots \emptyset]$ *is exactly* $C_j, 0 \leq j \leq l$.

Let the *state projection* of state $[p_0, p_1, \ldots, p_i, \ldots, p_l]$ into subcode index i be p_i if $p_i \neq \emptyset$ and empty if $p_i = \emptyset$. The *subcode projection* of T_v into subcode index i is defined by the symbol $|T_v|_i$ and consists of the subtrellis of T_v obtained by retaining all the non $\emptyset$ states in the state projection of the set of states into subcode index i and the edges between them. An overlayed trellis satisfies the property of *projection consistency* which stipulates that $|T_v|_i = T_i$. Thus every subtrellis T_j is embedded in T_v and can be obtained from it by a projection into the appropriate subcode index. We note here that the conventional trellis is equivalent to an overlayed trellis with $M/N = 1$.
Figure 3 shows the subtrellises for component codes C_0 and C_1 of the linear code defined earlier, overlayed to obtain the tailbiting trellis in Figure 4.

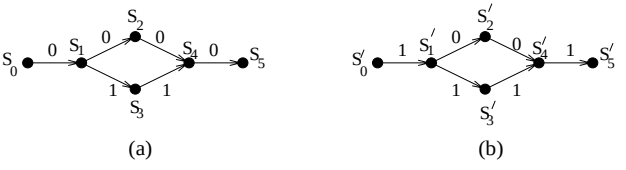

Fig. 3. Minimal trellises for (a) $C_0 = \{0000, 0110\}$ and (b) $C_1 = \{1001, 1111\}$

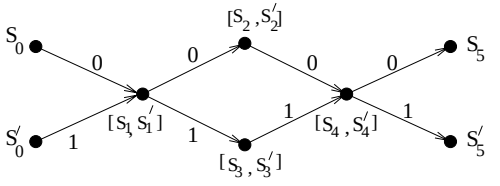

Fig. 4. Trellis obtained by overlaying trellis in figures 3(a) and 3(b)

It is shown that not all decompositions give overlayed trellises satisfying the specified bound on the width. Necessary and sufficient conditions for a decomposition to yield a tailbiting trellis with a specified bounded width are also given

in [6]. For purposes of decoding, we need the decomposition of the original conventional trellis into subtrellises that can be overlayed to form a tailbiting trellis. Each subtrellis has been shown to correspond to a coset of a group, and can be generated from the appropriate coset leader.

4 Decoding

Decoding refers to the process of forming an estimate of the transmitted codeword x from a possibly garbled version y. The received vector consists of a sequence of n real numbers where n is the length of the code. The soft decision decoding algorithm can be viewed as a shortest path algorithm on the trellis for the code. Based on the received vector, a cost $l(u, v)$ can be associated with an edge from node u to node v. The well known Viterbi decoding algorithm [20] is essentially a dynamic programming algorithm, used to compute a shortest path from the source to the goal node. Define a *winning* path as a shortest path from one of the start nodes to a final node. For a tailbiting trellis, decoding is complicated by the fact that non-accepting paths in the overlayed automaton share states with accepting paths. Thus a winning path at a goal node may not be an accepting path. We have designed and implemented a two phase algorithm which outputs a winning accepting path. The algorithm is outlined in [6] and described in detail along with proofs of correctness in [5]. We describe it informally here, along with a tiny example.

During the first phase, a Viterbi algorithm is run on the trellis and *survivors* i.e. shortest paths from any source node to all nodes are computed. Each node stores the cost of the survivor at itself at the end of the first phase. The second phase is an adaptation of the A^* algorithm[9], well known in the artificial intelligence community, and may be viewed as an adaptation of the Dijkstra algorithm with node to goal estimates added to source to node costs. All survivors at goal nodes are gathered at the end of the first phase. We term accepting paths as $s_i - f_i$ paths and non accepting paths as $s_i - f_j$ paths, with $i \neq j$. The second phase only needs to look at subtrellises T_j such that the winning path in T_j is an $s_i - f_j$ path and such that there are no $s_k - f_k$ paths with smaller cost,(we call such trellises, *residual* trellises) as the cost of a winning $s_i - f_j$ path will be an underestimate of that of the winning $s_j - f_j$ path. Any $s_i - f_j$ path with estimated cost greater than an $s_k - f_k$ path can therefore never be a winner. All residual trellises are candidates for the second phase. Decoding begins at the best candidate, i.e the one with the least estimate. The current estimates of all other residual trellises are stored in a heap. If at any instant the estimated cost of the current trellis exceeds the minimum value on the heap, the search in the current trellis is terminated, its estimate inserted into the heap, and the trellis corresponding to the minimum value in the heap taken up for searching next. Thus the algorithm makes its way towards the goal travelling on the best subtrellis seen so far at any given instant. As soon as the goal node is reached, we are sure that we have the

winning path. For high signal to noise ratios it is observed that the algorithm either does not need the second phase at all, or that it usually stays on a single subtrellis for the whole of the second phase. We illustrate with a tiny example. Though this one has only one residual trellis, it serves to illustrate the idea.

Figure 5 gives an overlayed trellis with some hypothetical costs. The nodes are labeled with survivors(within parentheses) after the first phase. Since the four codewords have costs 11,9,10,12, the winning path is $acefg$ with a cost of 9. The first phase outputs winners with cost 6 at g and 10 at h, corresponding to paths $bcefg$ and $bcefh$. Subtrellis corresponding to $s_i - f_j$ pair $b-g$ is a residual trellis so we begin decoding at a with estimate 6. At c the estimate changes to 4+(6-1)=9; at d it is 8+(6-5)=9; at e it is 5-(6-2)=9; at f it is min(9+(6-4), 7+(6-4))=9; at g it is 9+0=9. Hence the winning path is $acefg$.

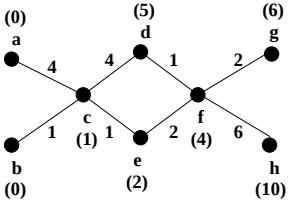

Fig. 5. Tailbiting trellis with hypothetical edge costs and survivors after first phase

5 Implementation

The package has been implemented in C.The minimal trellis construction algorithm of Kschischang-Sorokine that is implemented has been validated by generating minimal trellises for several codes for which these structures have been published, among them the (48,24) quadratic residue code, the (24,12) Golay code and the (16,7) lexicode(for which the state complexity profiles are available in [19].)

For the decomposition, subtrellises are generated from a tailbiting trellis by carrying out a forward traversal from each start state and keeping track of which nodes reachable from the start state also reach the appropriate final state. Thus the complexity of subtrellis generation from a tailbiting trellis is $O(s)$ where s is the number of states. Additional storage is not required for the subtrellises as each node of the tailbiting trellis has a vector of length $l+1$ associated with it (where $l+1$ is the number of subtrellises), which indicates whether a node of the tailbiting trellis is present in a certain subtrellis or not.

For the decoding, the channel is modeled as an AWGN channel using a random number generator that produces numbers that are normally distributed with mean 0 and variance $\frac{N_0}{2}$, where N_0 is the noise energy level. The tail-biting trellis is implemented as a two dimensional array of states reprepresenting vertices. Each state contains incoming and outgoing branches. Branches contain labels that correspond to a codeword symbols. If there are $l+1$ subtrellises, each state has storage for the $l+2$ surivivor paths (one obtained in the first phase and the remaining $l+1$ for each individual subtrellis to be used in the second phase), and for the corresponding metric. The metric along a branch is computed at runtime depending on the generated random codeword. Each state contain a membership array of length $l+1$ to indicate the membership in subtrellises. If vertex v belongs to subtrellis i, then the i'th bit of the array is set to 1, otherwise it is set to 0. The heap required in the second phase is implemented as an array.

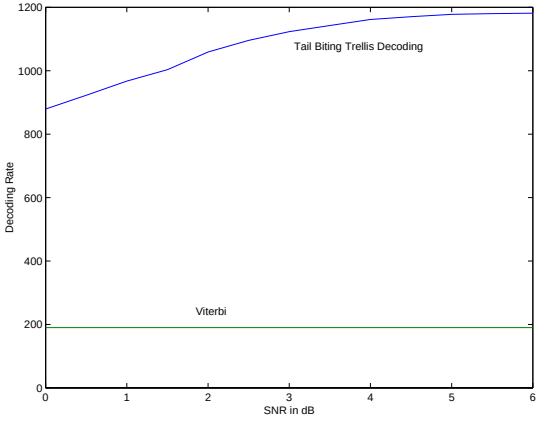

Fig. 6. Rates of decoding using the Viterbi algorithm and the two phase algorithm for the Golay code

5.1 Simulation Results

We present the simulation results of the algorithm tested on the extended (24,12) Golay Code. The rate of the code is $\frac{1}{2}$. The minimal tail-biting trellis of the 12-section (24,12) Golay code has a uniform state complexity of 16 at all time indices and is described in [3]. The conventional 12-section trellis of the (24,12) Golay Code has the state complexity profile (1,4,16,64,256,256,256,256,256,64,16,4,1). The conventional trellis has 1066 states and 2728 branches and the tail-biting trellis has 208 states and 384 branches. Each branch of the 12-sectioned Golay code represents 2 code bits.

A large number of random codewords is generated for various signal-to-noise ratios and the average rate of decoding is calculated by running the algorithm on

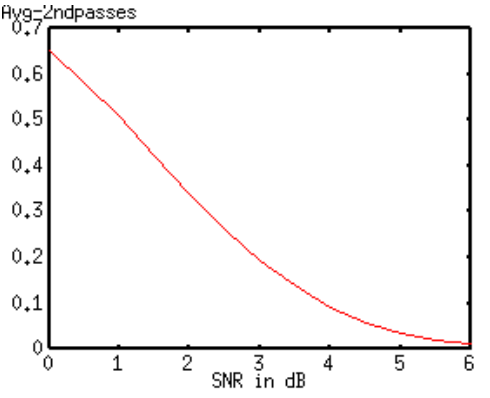

Fig. 7. Probability of a second pass in the two phase algorithm as a function of SNR

the tail-biting trellis. The results are compared with the Viterbi rate of decoding on the conventional trellis. The result is presented in figure 6. The rate graph shows that the two phase algorithm on the tailbiting trellis is significantly better than Viterbi decoding on the conventional trellis even at a low SNR of 0dB. While the Viterbi rate of decoding remains constant around 190 codewords/sec for SNR values in the range [0,6], the rate of the proposed algorithm increases steadily from 879 to 1181 codewords/sec. It is known[19] that the Viterbi algorithm on a trellis with V nodes and E edges requires $|E|$ multiplications and $|E| - |V| + 1$ additions. From the vertex and edge counts for the conventional and tailbiting trellises for the Golay code given above, we conclude that the overheads in heap operations and the number of switches are not significant.

It is also seen that for high SNR values, the decoding rarely needs a second phase. From figure 7 we see that the probability that the algorithm requires a second phase decreases from 0.652 to 0.012 as SNR increases from 0 to 6dB. During the second phase the algorithm switches from one subtrellis to other if there is a subtrellis on top of the heap with smaller metric than the subtrellis that is currently being expanded. Figure 8 shows that the average number of switches in the second pass decreases steadily to 1.53 showing that the search in second phase is usually restricted to a single subtrellis for large SNR values.

The measure of the probabilty of decoding error of a maximum likelihood decoder is given by the Bit Error Rate (BER). For a codeword C of an (n, k) code with each codeword symbol requiring m bits, mn bits are transmitted. If the decoder decodes to codeword C', whose mn bits differ from C in e locations, the bit error rate is $\frac{e}{mn}$. Thus, BER is the decoding error per bit. The bit error rate for the tail-biting trellis decoding is presented in figure 9. The probability of decoding error is very small for large SNR values, and the curve is consistent with others obtained in the literature for the Golay code .

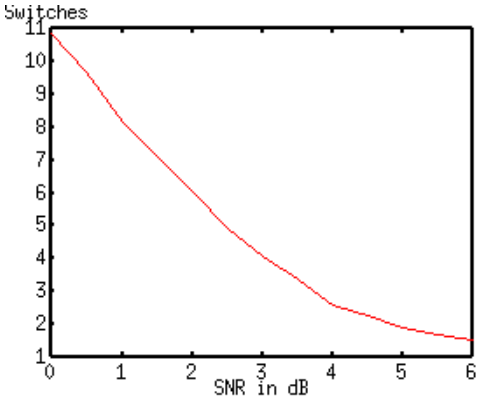

Fig. 8. Average number of switches between trellises when there is a second pass

6 Conclusion

We have implemented a package for the implementation of block codes as trellises and an efficient decoding algorithm and simulator for tailbiting trellises. Inclusion of a module for the conversion from conventional to tailbiting trellises when the full theory is available will make this, we hope, an useful general purpose tool for the coding community.

References

1. L.R.Bahl, J.Cocke, F.Jelinek, and J. Raviv, Optimal decoding of linear codes for minimizing symbol error rate, *IEEE Trans. Inform. Theory* **20**(2), March 1974, pp 284-287.
2. R.E. Blahut, *Theory and Practice of Error control Codes*, Addison Wesley, 1984.
3. 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.
4. G.C.Clark, Jr. and J.B. Cain, *Error-Correction Coding for Digital Communication.*, New York: Plenum, 1981.
5. Amitava Dasgupta, Priti Shankar, Kaustubh Deshmukh, and B.S,Rajan *On Viewing Block Codes as Finite Automata,* Technical Report IISc-CSA-99-7, Department of Computer Science and Automation, Indian Institute of Science, Bangalore-560012,
6. K.Deshmukh, Shankar,P., Dasgupta,A.,Sundar Rajan,B., On the many faces of block codes, in *Proceedings of STACS 2000, LNCS 1770* , (Lille, France, February 2000), pp 53-64.
7. G.D. Forney, Jr.,Coset codes II: Binary lattices and related codes, *IEEE Trans. Inform. Theory* **36**(5), Sept. 1988,pp 1152-1187.

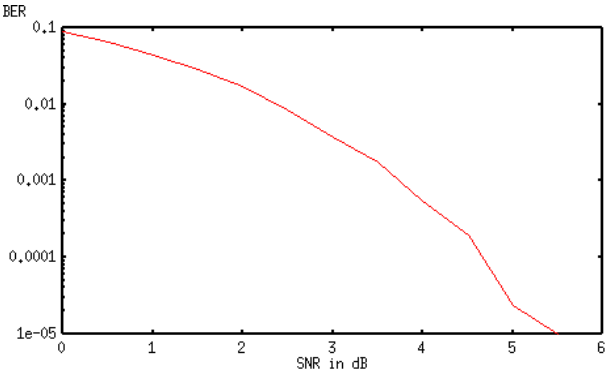

Fig. 9. Variation of the bit error rate with SNR

8. G.D. Forney, Jr. and M.D. Trott, The dynamics of group codes:State spaces, trellis diagrams and canonical encoders, *IEEE Trans. Inform. Theory* **39**(5) Sept 1993,pp 1491-1513.
9. 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.
10. Ralf Kotter and Vardy, A.,Construction of Minimal Tail-Biting Trellises,in *Proceedings IEEE Information Theory Workshop* (Killarney, Ireland, June 1998), 72-74.
11. Ralf Kotter and Vardy, A., The theory of tailbiting trellises, (manuscript in preparation).
12. F.R.Kschischang and V.Sorokine, On the trellis structure of block codes, *IEEE Trans. Inform. Theory* **41**(6), Nov 1995,pp 1924-1937.
13. D.Lind and M.Marcus, *An Introduction to Symbolic Dynamics and Coding*, Cambridge University Press, 1995.
14. F.J. MacWilliams and N.J.A. Sloane, *The Theory of Error Correcting Codes*, North-Holland, Amsterdam, 1981.
15. J.L.Massey, Foundations and methods of channel encoding, in *Proc. Int. Conf. on Information Theory and Systems* **65**(Berlin, Germany) Sept 1978.
16. R.J.McEliece, *The Theory of Information and Coding*, Encyclopedia of Mathematics and its Applications, Addison Wesley, 1977.
17. R.J.McEliece, On the BCJR trellis for linear block codes, *IEEE Trans. Inform. Theory* **42**,November 1996, pp 1072-1092.
18. D.J. Muder, Minimal trellises for block codes, *IEEE Trans. Inform. Theory* **34**(5), Sept 1988,pp 1049-1053.
19. A.Vardy, Trellis structure of codes, in *Handbook of Coding Theory*,V.S. Pless and W.C. Huffman, Eds.,Elsevier Science, 1998.
20. A.J. Viterbi, Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, *IEEE Trans. Inform. Theory* **13**, April 1967, pp 260-269.
21. N.Wiberg, H.-A. Loeliger and R.Kotter, Codes and iterative decoding on general graphs, *Eoro. Trans. Telecommun.*,**6** pp513-526, Sept 1995.
22. J.K. Wolf, Efficient maximum-likelihood decoding of linear block codes using a trellis, *IEEE Trans. Inform. Theory* **24** pp 76-80.