A Code Analysis Program *

George Ginis

June 2001

1 Introduction

A program for the analysis of linear codes is presented. The program is capable of producing several parameters characterizing the performance of a code. The underlying method is a forward search on the trellis of the code.

A linear code is defined as a code where the sum of any two codewords produces some other valid codeword. Convolutional codes and trellis codes belong to this category. On the other hand, "codes" defined by partial response (PR) channels are non-linear. The linearity property allows for the simplification of the search procedure, since one only needs to identify the error event paths with respect to some reference codeword (e.g. the all 0's codeword for convolutional codes). The error event paths with respect to any other codeword are easily derived by the error paths for the reference codeword.

The program produces a set of parameters useful for the estimation of the code performance:

- Minimum distance between trellis paths: d_{min}
- Number of neighboring paths at minimum distance: N_e
- Number of neighboring paths at next to minimum distances: N_1,N_2,N_3,...
- Number of bit errors corresponding to minimum distance error events: N_b
- Number of bit errors corresponding to next to minimum distance error events: $N_{b,1}$,$N_{b,2}$,$N_{b,3}$,...
- Maximum path length corresponding to a minimum distance error event: L_D
- Maximum path lengths corresponding to next to minimum distance error events: $L_{D,1}$,$L_{D,2}$,$L_{D,3}$,...

The number of parameters corresponding to error events for next to minimum distances can be determined by the user. Knowing the number of bit errors is mainly useful for the case of convolutional codes.

2 Algorithm description

As mentioned previously, the algorithm performs a forward search on the trellis, until either all the parameters of interest have been determined, or some bound on the number of forward steps has been exceeded.

A matrix L is maintained with rows corresponding to the number of trellis states, and columns corresponding to the parameters:

$$d_{min}, N_e, N_1, N_2, N_3, \ldots, N_b, N_{b,1}, N_{b,2}, N_{b,3}, \ldots$$

*Many thanks to Hichan Moon for the several helpful discussions, and the sharing of ideas contributing to the improvement of the algorithm.

This matrix stores information about paths starting from state 0, diverging at the first time instant, and eventually reaching the specified state, under the constraint that the intermediate states are different than 0. The entries contain information with respect to *paths having length equal to the total number of forward steps.* Each time the search progresses forward by one step, L is updated.

Additionally, a vector L_1 is maintained with information about paths starting from state 0, diverging at the first time instant, and reaching again state 0. The entries represent the parameters:

$$d_{min}, N_e, N_1, N_2, N_3, \ldots, N_b, N_{b,1}, N_{b,2}, N_{b,3}, \ldots, L_D, L_{D,1}, L_{D,2}, L_{D,3}, \ldots$$

However, the difference is that these parameters now refer to *all possible paths with length shorter than the number of forward steps.* As before, each time the search moves forward by one step, L_1 is updated.

The algorithm is first described in general terms: The first step, corresponding to the first "divergence" event on the trellis, involves the initialization of L. Then, forward "propagation" steps are performed, until the maximum number of steps has been exceeded, or some condition is satisfied indicating that all quantities of interest have already been determined, and that searching further is useless. At each such propagation step, L is updated for all states. Then, L_1 is also updated making use of the information stored in L regarding state 0. In other words, information regarding paths of length shorter than the current number of steps, that started at state 0, diverged, and then converged back to state 0 is updated by using the corresponding information for paths of length equal to the current number of steps. An example of the algorithm operation is shown in Figure 1.

It is worthwhile to discuss in more detail the methods used to compute the path parameters. For each state, a polynomial representation can be adopted to convey path information. In order to represent number of neighbors information, the polynomial is defined as:

$$g_n(x) = N_e x^{d_{min}} + N_1 x^{d_{min}+1} + N_2 x^{d_{min}+2} + N_3 x^{d_{min}+3} + \ldots$$

For representing number of bit errors information, the form of the polynomial becomes:

$$g_b(x) = N_b x^{d_{min}} + N_{b,1} x^{d_{min}+1} + N_{b,2} x^{d_{min}+2} + N_{b,3} x^{d_{min}+3} + \ldots$$

Observe that the exponents do not actually have to be integers and the exponent increments can be different than 1 (e.g. in trellis codes).

The advantage of this representation is the ease with which the "embedded" information can be updated. As an example, suppose that the algorithm discovers a new path of distance d with n_b bit errors. Then, the number of neighbors information is updated by adding the term x^d to $g_n(x)$, while the number of bit errors information is updated by adding the term $n_b x^d$ to $g_b(x)$. Also, since only a few lowest order terms are of actual interest, one needs to store only the first exponent and few coefficients, which will correspond to minimum distance and next to minimum distances error events.

Some more algorithmic details can now be provided: The vector L_1 is initialized to all 0's, except for the d_{min} entry, which is set to infinity (or some very large value). Similarly, the matrix L is also initialized to all 0's, except for the d_{min} entries of all states, which are set to infinity. For the first step, it is assumed that the initial state is 0. Then, for each possible input m_0 leading to some state $j \neq 0$, the distance $d_{0,j}$ and the number of bits in error n_{0,j,m_0} can be computed with respect

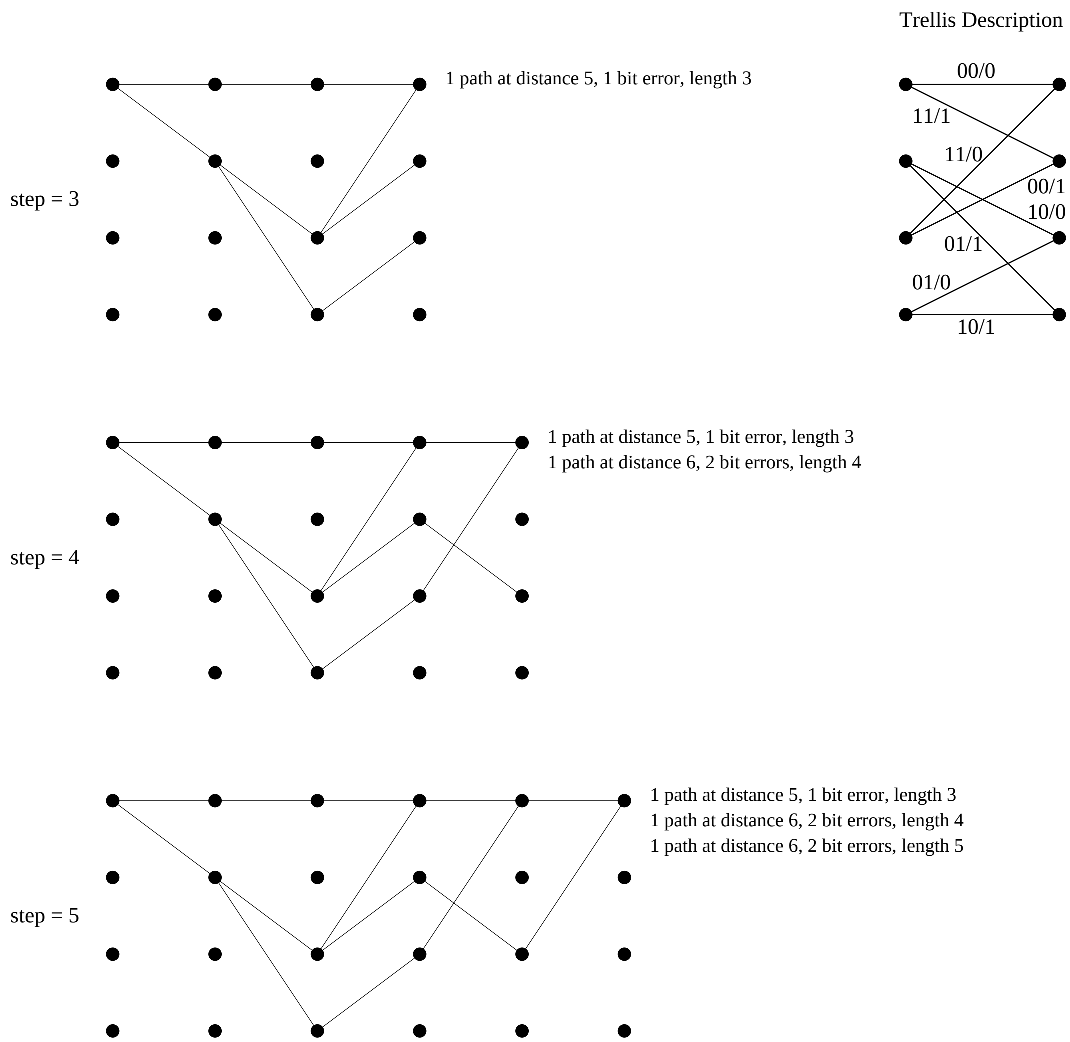

Figure 1: Example of the algorithm operation (only minimum and next to minimum distance paths are shown). As forward propagation progresses, more paths are discovered.

to the path leading to state 0. Each such input corresponds to a branch, and the relevant number of neighbors (or number of bit errors) information can be expressed as a monomial. Adding up all such monomials produces polynomials for each target state $j \neq 0$:

$$g_{n,j}(x) = \sum_{m_0 \in B(0,j)} x^{d_{0,j}}$$

and

$$g_{b,j}(x) = \sum_{m_0 \in B(0,j)} n_{0,j,m_0} x^{d_{0,j}}$$

where $B(0,j)$ is the set of inputs which lead from state 0 to state j. Accordingly, L is updated.

During the "propagation" step, L is first updated. For each state $i \neq 0$, the branches corresponding to each possible input m_i are considered. Assume that the initial polynomials for state i are $g_{n,i}(x)$ and $g_{b,i}(x)$, and that the branch leading from state i to state j with input m_i has cost $d_{i,j}$ and n_{i,j,m_i} bits in error. Then, the updated polynomials for any state j can be expressed as:

$$g_{n,j}^{new}(x) = \sum_{i \neq 0} \sum_{m_i \in B(i,j)} g_{n,i}(x) x^{d_{i,j}}$$

and

$$g_{b,j}^{new}(x) = \sum_{i \neq 0} \sum_{m_i \in B(i,j)} \left(g_{b,i}(x) + n_{i,j,m_i} g_{n,i}(x) \right) x^{d_{i,j}}$$

where $B(i,j)$ is the set of inputs which lead from state i to state j. Note that multiplying by $x^{d_{i,j}}$ implies a distance increase for all error events, which is the consequence of traversing a branch. Also, adding $g_{b,i}(x)$ with $n_{i,j,m_i} g_{n,i}(x)$ is performed to update the number of bits in error count. Thus, L is updated.

Once the new L has been determined, a new L_1 can be derived fairly easily. Suppose that $g_n(x)$, $g_b(x)$ are the original polynomials expressing the information in L_1, and $g_{n,0}(x)$, $g_{b,0}(x)$ are the polynomials expressing the information about state 0 in the recent L. Then, the entries of L_1 can be updated as dictated by the following expressions:

$$g_n^{new}(x) = g_n(x) + g_{n,0}(x)$$

and

$$g_b^{new}(x) = g_b(x) + g_{b,0}(x)$$

Thus, L_1 conveys information about all paths with lengths up to the current number of steps.

The maximum path length information can lastly be determined. If during the last update of L_1, the minimum distance changed, then all lengths are set to the current number of steps. Otherwise, if one of the $N_e, N_1, N_2, N_3, \ldots$ quantities of L_1 changed, then the corresponding distance (one of $L_D, L_{D,1}, L_{D,2}, L_{D,3}, \ldots$) is set to the current number of steps.

If during the update of L, it is observed that the d_{min} for some state i is significantly larger than the currently known d_{min} for state 0, then state i can be ignored for the polynomial calculations. If it is realized that *all* states satisfy this condition, then forward "propagation" can be terminated. The interpretation of "significantly larger" depends on how many next to minimum distances are of interest.

Finally, if the number of forward steps exceeds some preset upper bound, then again the algorithm terminates. This event may be an indication that the code is "catastrophic", which can be interpreted as the case where minimum distance error events have arbitrarily large path lengths.

3 How to use the program

The algorithm has been programmed in MATLAB. Although it is written for use with convolutional codes, it may easily be adapted to handle trellis codes. The program consists of the following modules:

- Main module, `dmin_main.m` .
- Module initializing the state information, `dmin_init.m` . It is called by the main module.
- Module performing a forward propagation on the trellis, `dmin_prop.m` . It is called by the main module.

- Function performing polynomial addition, `polyadd.m` .
- Function computing hamming distance, `bdistance.m` . Of course, this function must be changed when considering trellis codes.

A desciption of the trellis characterizing the code must be provided to the program. The main module uses the following arguments:

- Number of trellis states, `nstates` .
- Number of input bits, `ninput` .
- Matrix describing the trellis transitions, `trellis_transition` . The number of columns equals the number of possible inputs, and the number of rows equals the number of trellis states. Each row corresponds to a trellis state, and each column corresponds to an input. Thus, the (i, j) entry of the matrix is defined as the next trellis state, assuming that the previous state is $i - 1$ and the input is $j - 1$ (expressed as a decimal).
- Matrix describing the trellis outputs, `trellis_output` . The definition is similar to `trellis_transition`, however the entries refer to the outputs of the code.

For example, the 4-state code shown in Figure 1 has the following description:

```
nstates = 4
ninput = 1
trellis_transition=[1 2; 3 4; 1 2; 3 4]
trellis_output=[0 3; 2 1; 3 0; 1 2]
```

The main module then returns the parameters `dmin`, `Ne`, `N1`, `N2`, `N3`, `Nb`, `Nb1`, `Nb2`, `Nb3`, `LD`, `LD1`, `LD2`, `LD3` and the matrix `L`. The number of produced parameters (relevant to next to minimum distances) can be adjusted through the variable `ndepth` in `dmin_main.m` (the default value is 3).

Obviously, obtaining the trellis description is not always easy, especially for codes with a large number of states. In order to help with this task, the following functions are also provided:

- Function deriving the trellis representation of a non-systematic rate $1/n$ encoder, `trellis_gen.m` . It uses the code generator matrix as an argument.
- Function converting matrix with elements with octal representation to matrix with elements with decimal representation, `gen_convert.m` . It is called by `trellis_gen.m` .
- Function deriving the trellis representation of a systematic rate $2/3$ encoder, `trellis_par23.m` . It uses the parity matrix as an argument.
- Function deriving the trellis representation of a systematic rate $3/4$ encoder, `trellis_par34.m` . It uses the parity matrix as an argument.

- Function converting parity matrix with elements with octal representation to matrix with binary elements, `par_binconv.m` . It is called by `trellis_par23.m` and `trellis_par34.m` .
- Functions converting decimals into binary vectors, and binary vectors into decimals, `dec2binm.m`, `binm2dec.m` . They are called by `trellis_par23.m` and `trellis_par34.m` .

4 Tables for common convolutional codes

Using the code analysis program, results were generated for the convolutional codes defined in Tables 10.1 through 10.5 of the EE379B course reader (`http://www.stanford.edu/class/ee379b/reader.html`).

Tables 1, 3, and 5 give the generator matrices of maximum free distance codes with rates 1/2, 1/3 and 1/4. Tables 2, 4 and 6 give the parameters of the corresponding codes, where non-systematic implementations are assumed. Tables 7 and 9 give the parity matrices of maximum free distance codes with rates 2/3 and 3/4. Tables 8 and 10 give the parameters of the corresponding codes, where systematic implementations are assumed.

states	$g_{11}(D)$	$g_{12}(D)$
4	7	5
8	17	13
16	23	35
32	77	51
64	163	135
128	323	275
256	457	755
512	1337	1475
1024	2457	2355
2048	6133	5745
4096	17663	11271
8192	26651	36477
16384	46253	77361
32768	114727	176121
65536	330747	207225
131072	507517	654315

Table 1: Generator matrices for rate 1/2 maximum free distance convolutional codes.

states	d_{min}	N_e	N_1	N_2	N_3	N_b	$N_{b,1}$	$N_{b,2}$	$N_{b,3}$	L_D	$L_{D,1}$	$L_{D,2}$	$L_{D,3}$
4	5	1	2	4	8	1	4	12	32	3	5	7	9
8	6	1	3	5	11	2	7	18	49	5	7	9	11
16	7	2	3	4	16	4	12	20	72	8	12	13	15
32	8	2	3	8	15	4	11	36	83	8	15	14	21
64	10	12	0	53	0	46	0	332	0	16	-	26	-
128	10	1	6	13	20	6	22	56	136	14	16	21	24
256	12	10	9	30	51	40	33	196	357	18	26	27	32
512	12	1	8	8	31	2	36	34	187	11	18	20	23
1024	14	19	0	80	0	82	0	530	0	22	-	30	-
2048	14	1	10	25	46	4	46	156	366	19	25	31	35
4096	15	2	10	29	55	6	46	177	386	18	27	30	35
8192	16	5	15	21	56	26	91	128	416	28	28	35	38
16384	17	3	16	44	62	17	78	276	484	27	34	41	41
32768	18	5	15	45	86	26	95	308	650	37	31	38	39
65536	19	9	16	48	112	55	80	344	914	33	33	43	42
131072	20	6	31	58	125	30	183	472	1115	27	34	43	47

Table 2: Parameters for rate 1/2 maximum free distance non-systematic convolutional codes.

states	$g_{11}(D)$	$g_{12}(D)$	$g_{13}(D)$
4	5	7	7
8	15	13	17
16	25	33	37
32	71	65	57
64	171	165	133
128	365	353	227
256	561	325	747
512	1735	1063	1257
1024	3645	2133	3347
2048	6531	5615	7523
4096	13471	15275	10637
8192	32671	27643	22617
16384	47371	51255	74263
32768	151711	167263	134337
65536	166255	321143	227277

Table 3: Generator matrices for rate 1/3 maximum free distance convolutional codes.

states	d_{min}	N_e	N_1	N_2	N_3	N_b	$N_{b,1}$	$N_{b,2}$	$N_{b,3}$	L_D	$L_{D,1}$	$L_{D,2}$	$L_{D,3}$
4	8	2	0	5	0	3	0	15	0	4	-	7	-
8	10	3	0	2	0	6	0	6	0	6	-	7	-
16	12	5	0	3	0	12	0	12	0	8	-	9	-
32	13	1	3	6	4	1	8	26	20	6	10	13	16
64	15	3	3	6	9	7	8	22	44	11	12	13	16
128	16	1	5	2	6	2	13	6	26	10	12	12	16
256	17	1	2	6	7	1	4	20	34	9	11	14	18
512	20	7	0	19	0	19	0	97	0	15	-	21	-
1024	21	4	1	4	14	12	2	16	64	16	12	18	19
2048	22	3	0	9	0	8	0	35	0	15	-	22	-
4096	24	2	8	10	15	6	30	44	71	17	23	23	25
8192	26	7	0	23	0	19	0	113	0	20	-	30	-
16384	27	6	4	6	21	22	18	28	112	24	25	25	28
32768	28	1	6	5	17	2	24	24	103	17	23	24	28
65536	28	1	0	10	0	4	0	44	0	22	-	27	-

Table 4: Parameters for rate 1/3 maximum free distance non-systematic convolutional codes.

states	$g_{11}(D)$	$g_{12}(D)$	$g_{13}(D)$	$g_{14}(D)$
4	7	7	7	5
8	17	15	13	13
16	37	35	33	25
32	73	65	57	47
64	163	147	135	135
128	367	323	275	271
256	751	575	633	627
512	0671	1755	1353	1047
1024	3321	2365	3643	2277
2048	7221	7745	5223	6277
4096	15531	17435	05133	17627
8192	23551	25075	26713	37467
16384	66371	50575	56533	51447
32768	176151	123175	135233	156627
65536	247631	264335	235433	311727

Table 5: Generator matrices for rate 1/4 maximum free distance convolutional codes.

states	d_{min}	N_e	N_1	N_2	N_3	N_b	$N_{b,1}$	$N_{b,2}$	$N_{b,3}$	L_D	$L_{D,1}$	$L_{D,2}$	$L_{D,3}$
4	10	1	1	1	3	2	1	4	9	4	4	7	6
8	13	2	1	0	3	4	2	0	10	6	5	-	8
16	16	4	0	2	0	8	0	7	0	7	-	8	-
32	18	3	0	5	0	6	0	17	0	8	-	11	-
64	20	10	0	0	0	37	0	0	0	16	-	-	-
128	22	1	4	3	2	2	10	10	8	9	11	13	12
256	24	1	3	4	7	2	9	14	29	10	13	14	16
512	26	3	0	4	0	6	0	10	0	12	-	13	-
1024	28	4	0	5	0	9	0	14	0	16	-	18	-
2048	30	4	0	4	0	9	0	14	0	15	-	18	-
4096	32	4	0	9	0	13	0	30	0	17	-	21	-
8192	34	1	0	11	0	3	0	38	0	18	-	22	-
16384	37	3	5	6	10	7	18	26	44	19	22	24	23
32768	39	5	7	10	4	17	26	48	18	21	26	25	24
65536	41	3	7	7	10	7	24	33	46	20	24	27	26

Table 6: Parameters for rate 1/4 maximum free distance non-systematic convolutional codes.

states	$h_1(D)$	$h_2(D)$	$h_3(D)$
8	17	15	13
16	05	23	27
32	53	51	65
64	121	113	137
128	271	257	265
256	563	601	475
512	1405	1631	1333
1024	1347	3641	3415
2048	5575	7377	4033
4096	14107	13125	11203
8192	20535	31637	27773
16384	66147	41265	57443
32768	100033	167575	155377
65536	353431	300007	267063

Table 7: Parity matrices for rate 2/3 maximum free distance convolutional codes.

states	d_{min}	N_e	N_1	N_2	N_3	N_b	$N_{b,1}$	$N_{b,2}$	$N_{b,3}$	L_D	$L_{D,1}$	$L_{D,2}$	$L_{D,3}$
8	4	1	5	24	71	3	15	98	331	2	4	7	9
16	5	7	23	59	240	26	91	276	1329	7	9	11	15
32	6	9	19	80	276	38	94	432	1711	9	11	15	19
64	6	1	17	47	145	6	78	241	865	5	13	15	18
128	7	6	26	105	378	27	141	635	2528	11	15	19	22
256	8	8	40	157	598	44	226	1034	4323	9	18	22	30
512	8	1	20	75	243	6	120	488	1815	7	15	20	22
1024	9	9	45	166	593	56	313	1199	4755	15	19	26	30
2048	10	29	0	473	0	200	0	3759	0	18	-	29	-
4096	10	4	34	127	450	27	248	1010	3848	14	19	24	27
8192	11	9	72	222	824	66	564	1949	7652	16	27	27	36
16384	12	58	0	847	0	469	0	7857	0	26	-	31	-
32768	12	25	0	462	0	193	0	4258	0	23	-	30	-
65536	12	7	26	120	437	56	227	1098	4344	22	25	33	37

Table 8: Parameters for rate 2/3 maximum free distance systematic convolutional codes.

states	$h_1(D)$	$h_2(D)$	$h_3(D)$	$h_4(D)$
32	45	71	63	51
64	151	121	177	111

Table 9: Parity matrices for rate 3/4 maximum free distance convolutional codes.

states	d_{min}	N_e	N_1	N_2	N_3	N_b	$N_{b,1}$	$N_{b,2}$	$N_{b,3}$	L_D	$L_{D,1}$	$L_{D,2}$	$L_{D,3}$
32	5	9	47	218	1121	34	203	1146	6634	7	9	15	17
64	6	30	112	640	3109	125	597	3715	20787	11	13	20	22

Table 10: Parameters for rate 3/4 maximum free distance systematic convolutional codes.