```
 1 /*
 2    RA.c
 3                (c) DJCM 98 09 28
 4
 5    Repeat-accumulate code simulator
 6
 7  read in code definition
 8  loop {
 9    encode source string
10    add noise
11    decode
12  }
13
14  Code definition: (stored in "alist")
15
16        Use of alist allows arbitrary numbers of repetitions
17        of each bit.
18
19   K                       source block length
20   n_1 n_2 ... n_K         number of repetitions of each source bit
21   N = sum n_k
22   alist  defines          permutation of N encoded bits
23                           note, an additional permutation of the N
                             accumulated
24        bits may be a good idea. (for non-memoryless channels)
25
26  transmitted bits are integral of encoded bits
27
28  Future plans:
29   clump source bits into clumps. Have multiple parallel accumulated
     streams.
30   Have little sub-matrices (like GF(q) ) defining response of
     accumulator to
31   clumps.
32
33 */
34
35 #include "./ansi/r.h"
36 #include "./ansi/rand2.h"
37 #include "./ansi/mynr.h"
38 #include "./ansi/cmatrix.h"
39
40 #include "./RA.h" /* this defines  data_creation_param ; RA_control */
41
42 int RA_encode ( unsigned char * , RA_control * , unsigned char * ) ;
43 static int t_to_b (  unsigned char * ,  RA_control * ) ;
44 int RA_decode ( RA_control * ) ;
45 int RA_horizontal_pass ( RA_control * ) ;
46
47 static void dc_defaults ( RA_control * ) ;
48 static int  process_command ( int , char **  , RA_control * ) ;
49 static void print_usage ( char ** , FILE * ,  RA_control *  );
50 static int  make_sense ( RA_control * ) ;
51 static int  make_space ( RA_control * ) ;
52 static int  score ( RA_control *  ) ;
53
54 static void finalline (  FILE * , RA_control * , int ) ; /* int = 1 to
   get loads of info */
```

```
 55
 56 static double bern ( int  , int  , double * , double * ,  double * ,
    double );
 57 static void histo ( FILE * ,  RA_control * ) ;
 58 static void snappyline ( RA_control * ) ;
 59 static void RA_free ( RA_control * ) ;
 60 static int  check_alist_MN ( alist_matrix * , RA_control * ) ;
 61 static double h2 ( double ) ;
 62
 63 void   main ( int , char ** ) ;
 64
 65 /*
 66         MAIN
 67                       */
 68 void main ( int argc, char *argv[] )
 69 {
 70   FILE   *fp  ;
 71
 72   int k ;
 73   RA_control         c ;
 74   dc_defaults   ( &c ) ;
 75
 76   if ( process_command (argc, argv, &c ) < 0 ) exit (0) ;
 77   if ( read_allocate_alist ( &(c.a) , c.afile ) < 0 ) exit (0) ;
 78   if ( check_alist_MN ( &(c.a) , &c ) < 0 ) exit (0) ;
 79   if ( make_sense ( &c  ) < 0 ) exit (0) ;
 80
 81   fprintf(stderr,"RA N=%d, K=%d, x=%6.3g xass=%6.3g fn=%6.3g
      fnass=%6.3g\n",
 82     c.N , c.K , c.gcx , c.gcxass , c.fn , c.fnass ) ;
 83   fflush(stderr);
 84
 85   if ( make_space ( &c ) < 0 ) exit (0) ;
 86
 87   if ( c.writelog ) {
 88     fp = fopen ( c.logfile , "w" ) ;
 89     if ( !fp ) {
 90       fprintf ( stderr , " couldn't open logfile %s\n" , c.logfile ) ;
 91       c.writelog = 0 ;
 92     } else fclose (fp ) ;
 93   }
 94
 95   if ( c.writelog ) {
 96     fp = fopen ( c.logfile , "w" ) ;
 97     if ( !fp ) {
 98       fprintf ( stderr , " couldn't open logfile %s\n" , c.logfile ) ;
 99       c.writelog = 0 ;
100     } else fclose (fp ) ;
101   }
102
103   if ( c.error_log ) {
104     fp = fopen ( c.error_logfile , "w" ) ;
105     if ( !fp ) {
106       fprintf ( stderr , " couldn't open logfile %s\n" , c.error_logfile
        ) ;
107       c.error_log = 0 ;
108     } else fclose (fp ) ;
109   }
```

```
110
111   /*
112       MAIN LOOP
113                   */
114
115   ran_seed ( c.vseed ) ;
116   c.message = 1 ;
117   for ( ;       ( c.message <= c.MESSAGE ) &&
118     ( ( c.failures==0 ) || ( c.failcount < c.failures ) ) ;
119   c.message ++ ) {
120
121     snappyline( &c ) ;
122     /* force parity bit at end */
123     c.sourceweight = random_cvector ( c.s , c.fs , 1 , c.K ) ;
124
125     if ( c.verbose > 2) {
126       printf ( "source vector:\n" ) ;
127       for ( k = 1 ; k <= c.K ; k ++ ) {
128   if ( c.s[k] ) printf ("1 ") ; else printf ( "0 " );
129       }
130       printf ( "\n" ) ;
131     }
132     RA_encode ( c.s , &c , c.t ) ;
133     if ( c.verbose > 2) {
134       printf ( "transmitted vector:\n" ) ;
135       for ( k = 1 ; k <= c.N ; k ++ ) {
136   if ( c.t[k] ) printf ("1 ") ; else printf ( "0 " );
137       }
138       printf ( "\n" ) ;
139     }
140     c.flipped = t_to_b ( c.t , &c ) ;
141     if ( c.verbose > 2 ) {
142       printf ( "received likelihoods:\n" ) ;
143       for ( k = 1 ; k <= c.N ; k ++ ) {
144   printf ("%1d " , (int) ( c.bias[k][1] * 10.0 ) ) ;
145       }
146       printf ( "\n" ) ;
147       for ( k = 1 ; k <= c.N ; k ++ ) {
148   printf ("%1d " , (int) ( c.bias[k][1] * 2.0 ) ) ;
149       }
150       printf ( "\n" ) ;
151     }
152     RA_decode ( &c ) ;
153
154     if ( score ( &c ) < 0 ) exit ( 0 ) ;
155     if ( c.verbose > 0 ) finalline ( stdout , &c , 0 ) ;
156     if ( c.printout ) { /* append */
157       fp = fopen ( c.outfile , ((c.outappend)? "a":"w") ) ;
158       if( !fp ) {
159   fprintf( stderr, "No such file: %s\n", c.outfile ) ;
160   finalline ( stderr , &c , 0 ) ;
161       }      else   {
162   finalline ( fp ,  &c , 0 ) ;
163   fclose ( fp ) ;
164       }
165     }
166     if (  (!( ( c.message+1 <= c.MESSAGE ) &&
167     ( ( c.failures==0 )
```

```
168       || ( c.failcount < c.failures ) ) ) )
169     || (!(c.message % c.big_write_period ) ) ) {
170       if ( c.printtot ) { /* write */
171   fp = fopen ( c.totoutfile , "w" ) ;
172   if( !fp ) {
173     fprintf( stderr, "No such file: %s\n", c.totoutfile ) ;
174     finalline ( stderr , &c , 1 ) ; /* totalline */
175   }      else   {
176     finalline ( fp , &c , 1 ) ;
177     fclose ( fp ) ;
178   }
179       }
180       if ( c.printhisto && c.block_valid ) { /* update histogram file */
181   fp = fopen ( c.histofile , "w" ) ;
182   if( !fp ) {
183     fprintf( stderr, "No such file: %s\n", c.histofile ) ;
184   }      else   {
185     histo ( fp , &c ) ;
186     fclose ( fp ) ;
187   }
188       }
189     }
190   }
191   snappyline( &c ) ; printf("\n") ;
192   RA_free (  &c ) ;
193 }
194
195 static void histo ( FILE *fp ,  RA_control *c ) {
196   int l;
197   double t , cum = 0.0 ;
198   double  tot = (double) c->block_valid ;
199
200   fprintf ( fp , "# total valid blocks %d\n" , c->block_valid ) ;
201   for ( l = 1 ; l <= c->loops ; l ++ ) {
202     t= (double) c->histo[l] ;
203     cum += t ;
204     fprintf ( fp , "%d\t%d\t%d\t%9.4g\t%9.4g\n" , l , (int)(t) ,
      (int)(cum) ,
205         t/tot , cum/tot ) ;
206   }
207
208 }
209 static void snappyline ( RA_control *c ) {
210   printf ( "%d:%du%dd%dl/%d\t" , c->block_errs , c->block_undet,
    c->block_det, c->block_detlw , c->message - 1 ) ; fflush ( stdout ) ;
211
212 }
213
214 /*                                                  <<<<N>>>>
215   Encoding method:
216   source bits d[1]..d[K] are mapped via an alist  ^ 1    1 1
217   into a pre-transmission vector                  K  1 1  1
218   s[1]..s[N] .                                    V   1 1   1
219
220   t[n] = t[n-1] ^ s[n]
221
222      s[n]      s[n+1]
223       |         |
```

```
224 0 .. -+-> t[n] -+-> t[n+1] ..... t[N]
225            |        |            |
226            V        V    V
227           y[n]      y[n+1]   y[N]
228
229   */
230
231 int RA_encode ( unsigned char *d , RA_control *c , unsigned char *t ) {
232   int n , k ; int status = 0 ;
233   alist_transpose_cvector_sparse_mod2 ( &c->a , d , t ) ; /* here 't'
      doubles as 's' */
234   /* accumulate */
235
236   if ( c->verbose > 2) {
237     printf ( "extended source vector:\n" ) ;
238     for ( k = 1 ; k <= c->N ; k ++ ) {
239       if ( t[k] ) printf ("1 ") ; else printf ( "0 " );
240     }
241     printf ( "\n" ) ;
242   }
243
244   for ( n = 2 ; n <= c->N ; n ++ ) {
245     t[n] = t[n]^t[n-1] ;
246   }
247
248   if ( c->verbose > 2) {
249     printf ( "accumulated transmission:\n" ) ;
250     for ( k = 1 ; k <= c->N ; k ++ ) {
251       if ( t[k] ) printf ("1 ") ; else printf ( "0 " );
252     }
253     printf ( "\n" ) ;
254   }
255   return status ;
256 }
257
258 /*
259    The channel outputs a normalized likelihood vector
260    bias[n] = P( yn | tn = 1 )
261
262    The state of the decoder is  q[1..N][0/1] and r[1..N][0/1]
263
264    q[n][s] = pseudoprior( s[n] = s )     s=0/1      initially 0.5
265
266    Use f/b algorithm to find:
267                                          initial conditions:
268    f[n][t] = P( y1...yn , tn=t )         f[0][0] = 1 ; f[0][1] = 0 ;
269    b[n][t] = P( yn...yN | tn=t )         b[N+1][0] = 1 ; b[N+1][1] = 1 ;
270
271    Using
272    f[n][t] = bias[n][t] * sum_{t': t'+s=t} ( f[n-1][t'] pi[n][s] )
273    b[n][t] = bias[n][t] * sum_{t': t'+s=t} ( b[n+1][t'] pi[n+1][s] )
274
275    Find likelihood contribution at n:
276       r[n][s] `=' P(y1..yN|s[n]=s) = sum_{s: t'+s=t} f[n-1][t] b[n][t']
277
278    Then in the vertical step we visit each incarnation of the source bit
279    for( r = 1 .. repetitions[k] ) (number on mlist)  n=nlist[r]
280       d[r][s] = d[r-1][s] * r[n][s] ;
```

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.