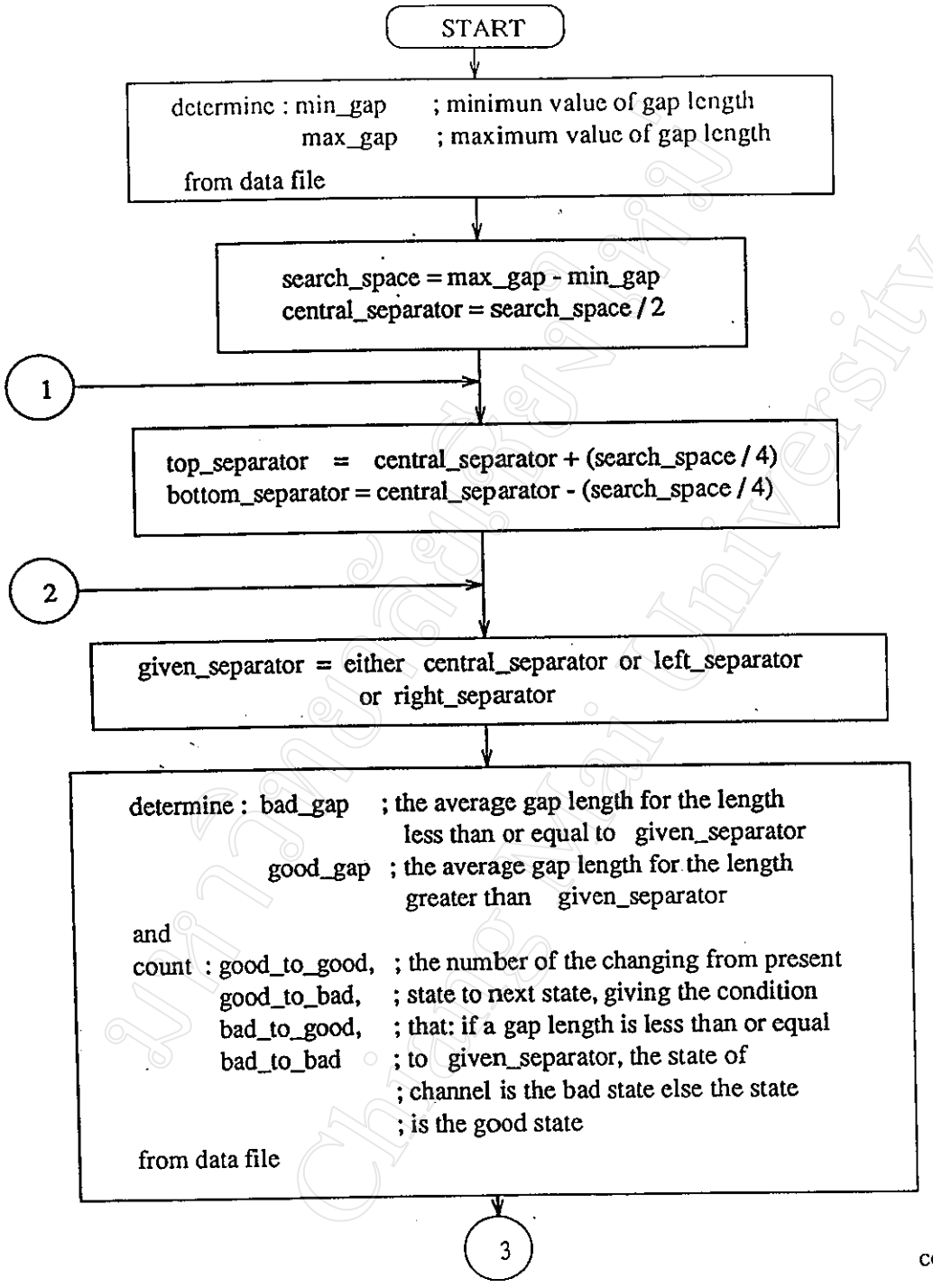
ภาคผนวก ก

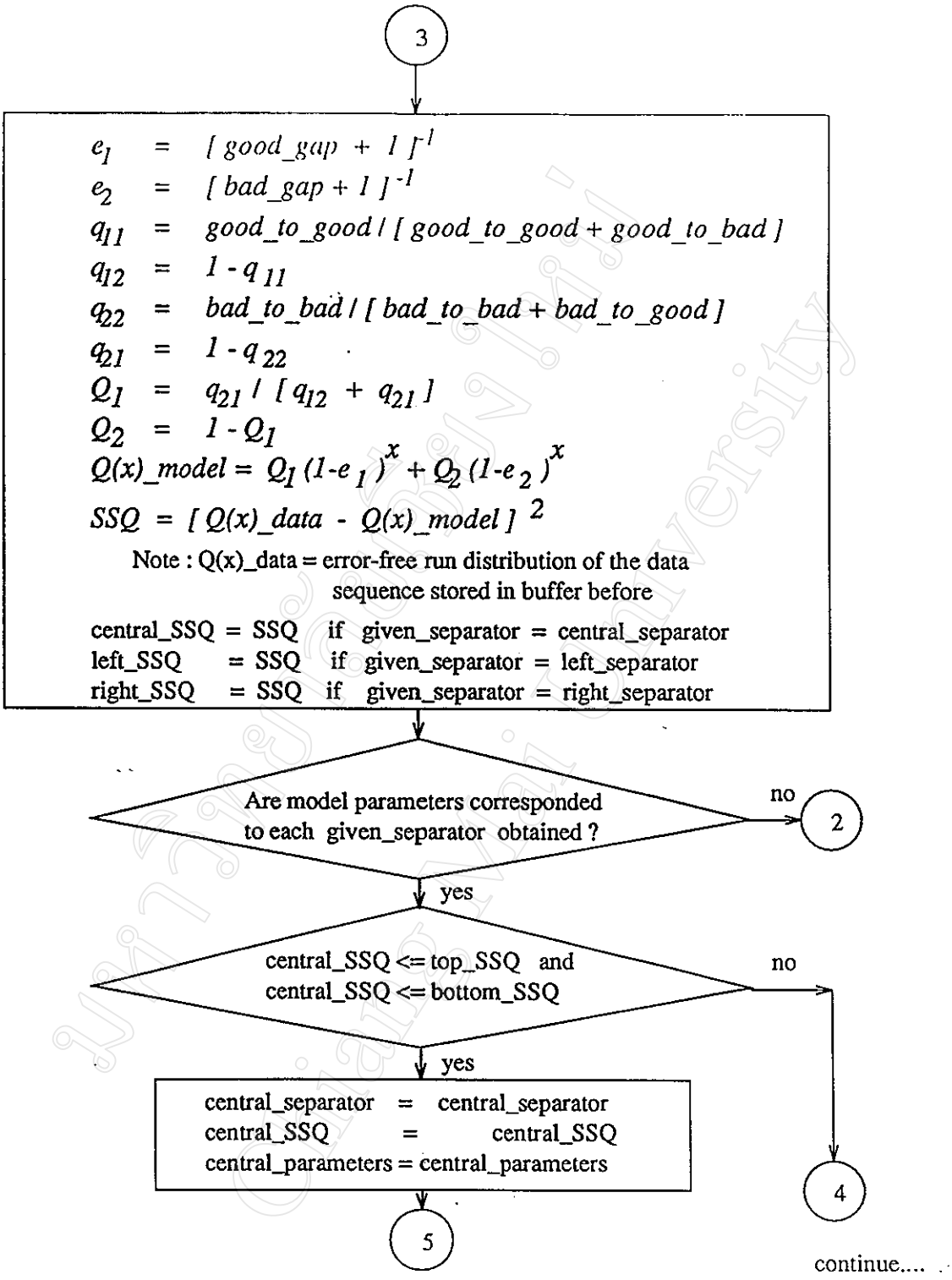
ผังแสดงขั้นตอนวิธีการหาค่าพารามิเตอร์ของ
แบบจำลองลูกโซ่มาร์คอฟสองสถานะ

มหาวิทยาลัยเชียงใหม่
Chiang Mai University

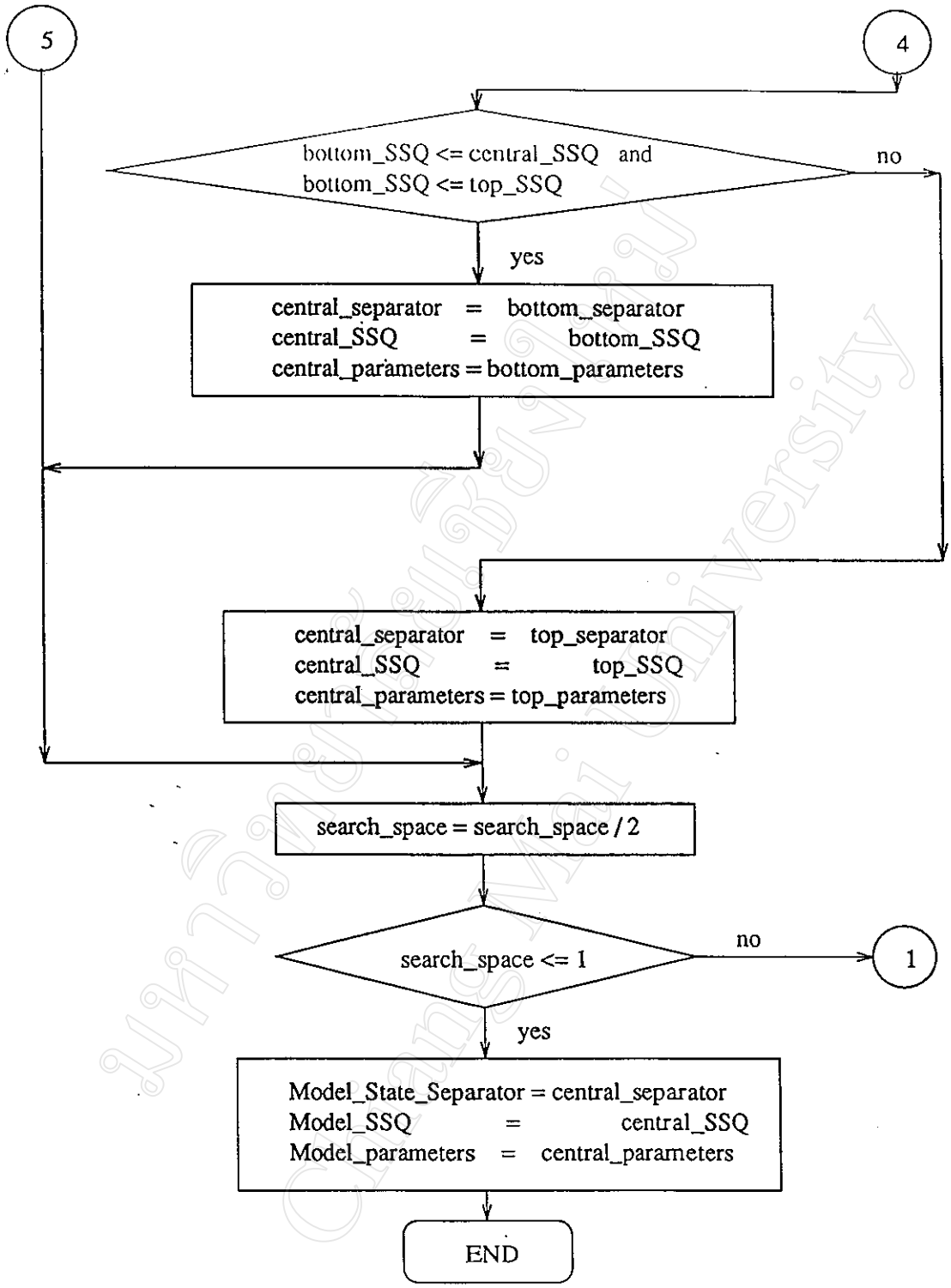
Optimisation using Binary Search of State Separator



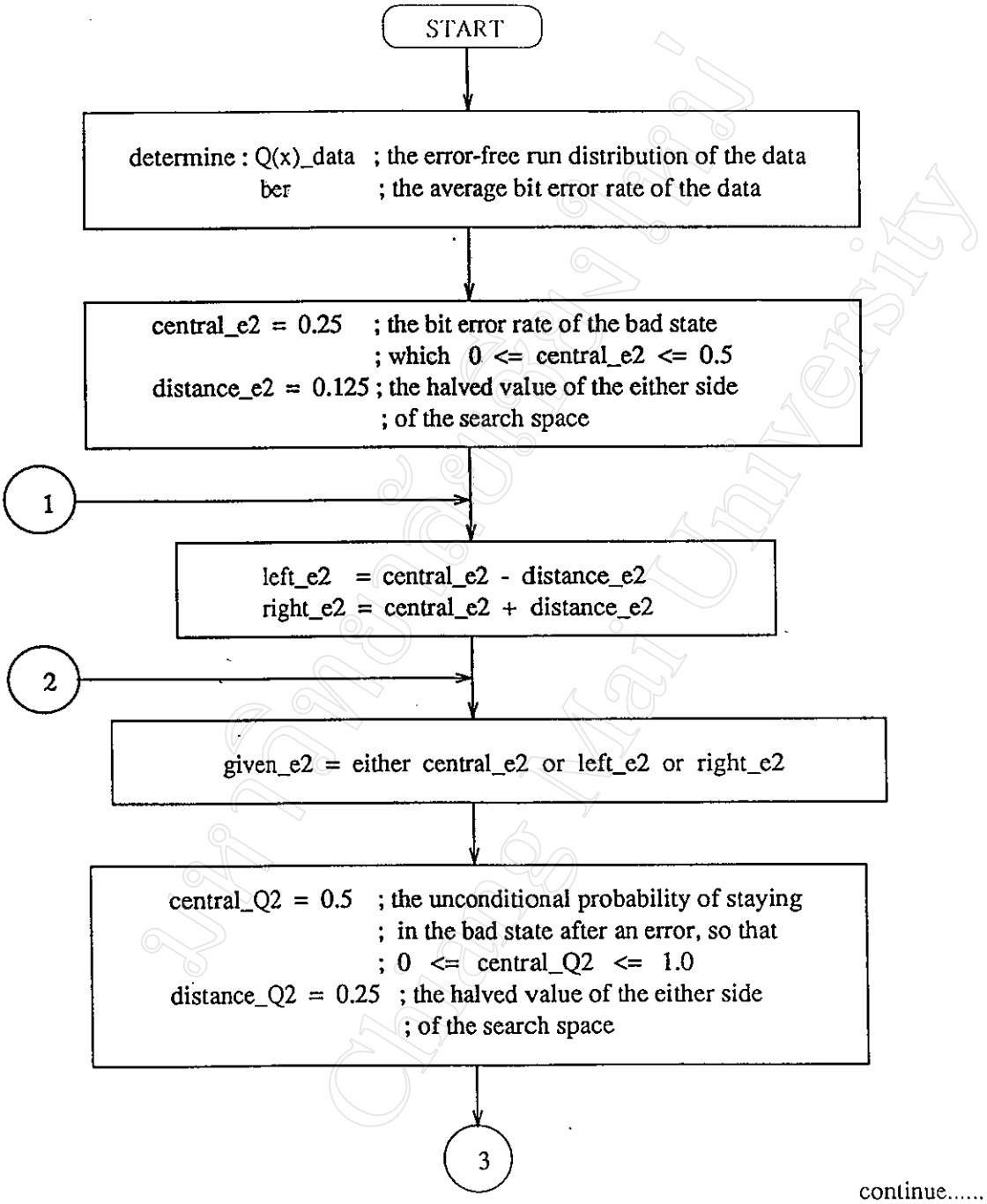
continue....


3
$e_1 = [\ good\_gap + 1\ ]^{-1}$
$e_2 = [\ bad\_gap + 1\ ]^{-1}$
$q_{11} = good\_to\_good / [\ good\_to\_good + good\_to\_bad\ ]$
$q_{12} = 1 - q_{11}$
$q_{22} = bad\_to\_bad / [\ bad\_to\_bad + bad\_to\_good\ ]$
$q_{21} = 1 - q_{22}$
$Q_1 = q_{21} / [\ q_{12} + q_{21}\ ]$
$Q_2 = 1 - Q_1$
$Q(x)\_model = Q_1 (1-e_1)^x + Q_2 (1-e_2)^x$
$SSQ = [\ Q(x)\_data - Q(x)\_model\ ]^2$
Note : Q(x)_data = error-free run distribution of the data sequence stored in buffer before
central_SSQ = SSQ if given_separator = central_separator
left_SSQ = SSQ if given_separator = left_separator
right_SSQ = SSQ if given_separator = right_separator
Are model parameters corresponded to each given_separator obtained ?
no
2
yes
central_SSQ <= top_SSQ and central_SSQ <= bottom_SSQ
no
yes
central_separator = central_separator
central_SSQ = central_SSQ
central_parameters = central_parameters
4
5

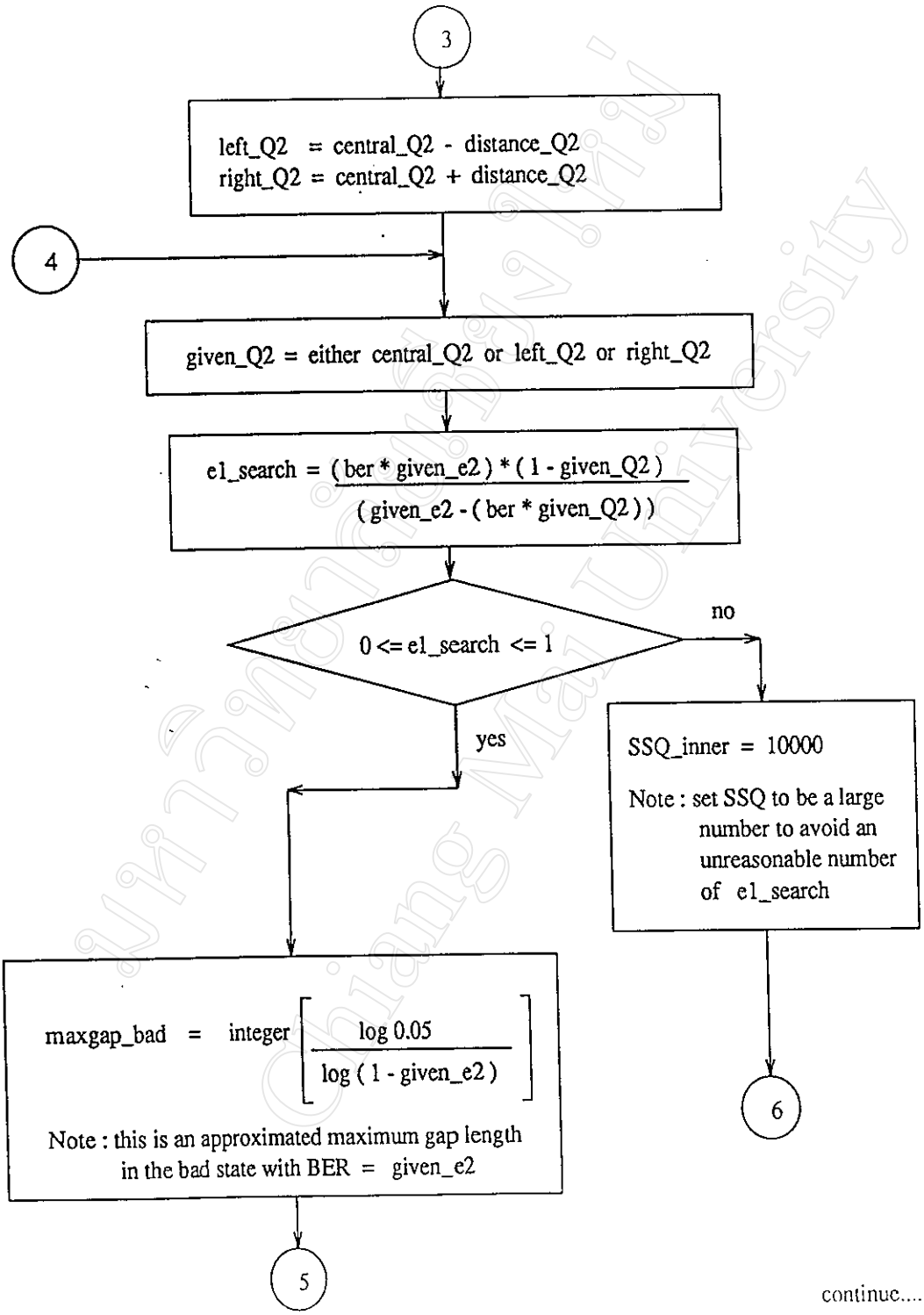
continue....


5
4
bottom_SSQ <= central_SSQ and
bottom_SSQ <= top_SSQ
no
yes
central_separator = bottom_separator
central_SSQ = bottom_SSQ
central_parameters = bottom_parameters
central_separator = top_separator
central_SSQ = top_SSQ
central_parameters = top_parameters
search_space = search_space / 2
search_space <= 1
no
1
yes
Model_State_Separator = central_separator
Model_SSQ = central_SSQ
Model_parameters = central_parameters
END

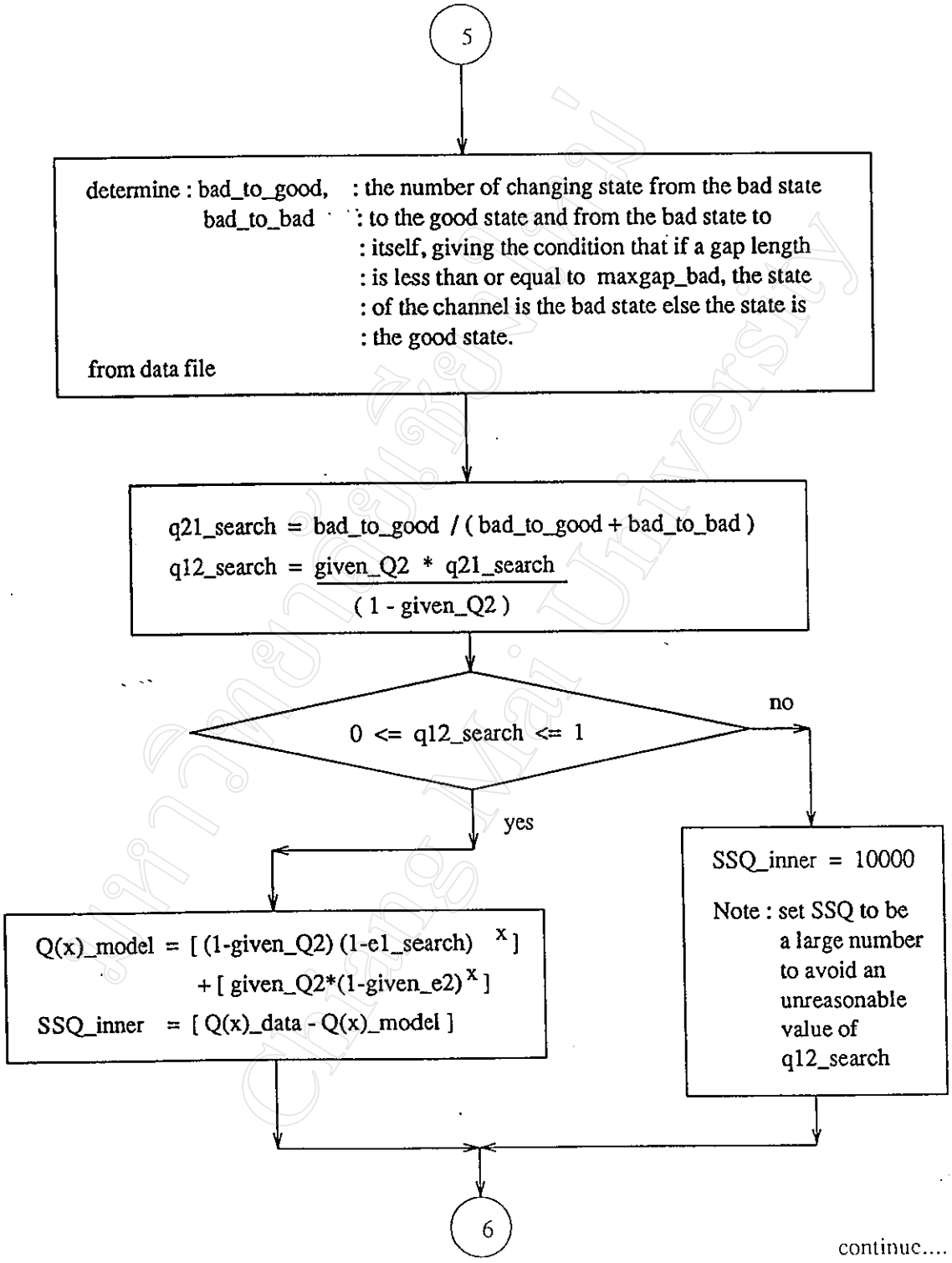
Optimisation using Binary Search of Bad State Parameters



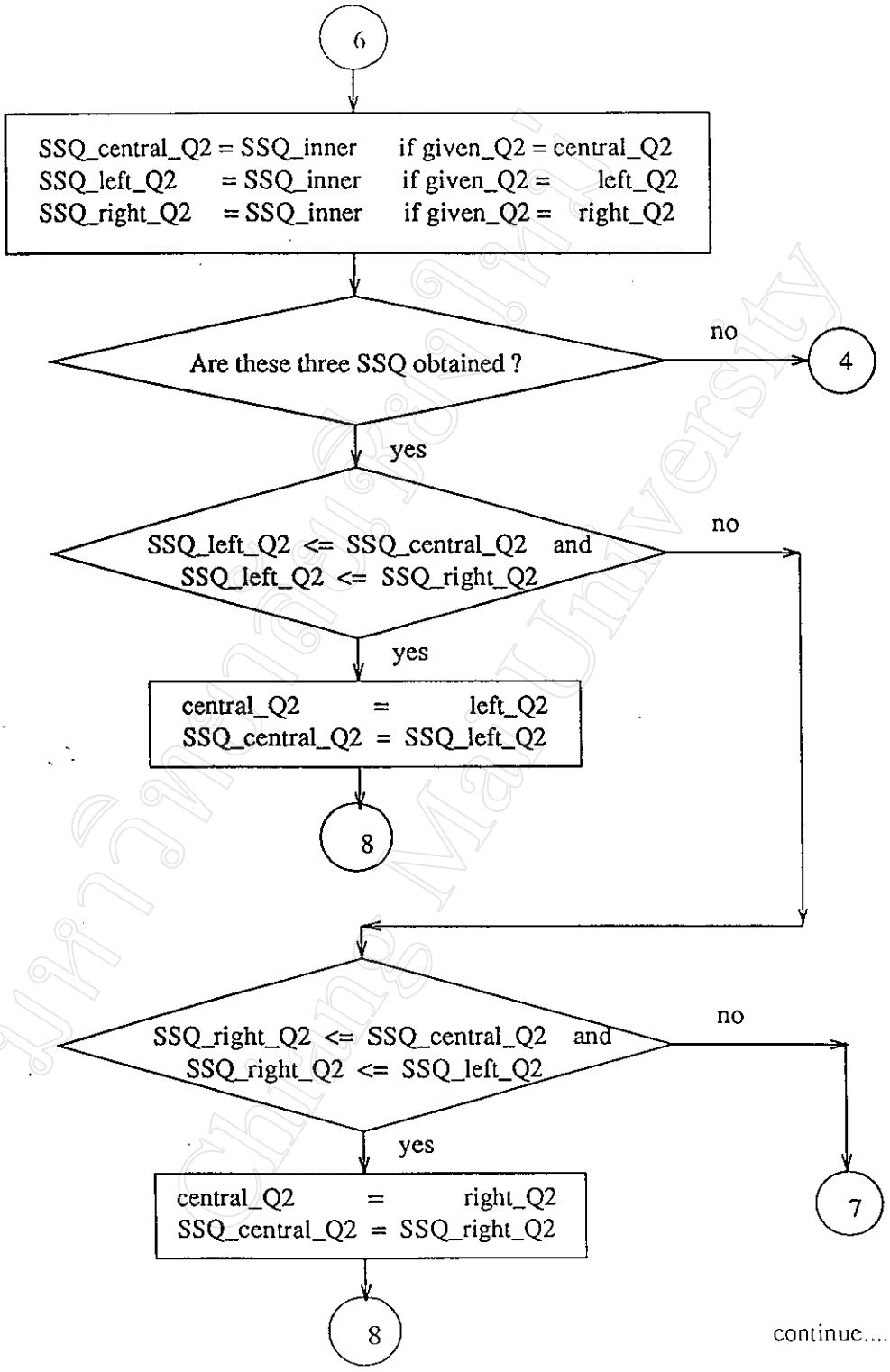
continue.....


3
left_Q2 = central_Q2 - distance_Q2
right_Q2 = central_Q2 + distance_Q2
4
given_Q2 = either central_Q2 or left_Q2 or right_Q2
e1_search = (ber * given_e2) * (1 - given_Q2) / (given_e2 - (ber * given_Q2))
0 <= e1_search <= 1
no
yes
SSQ_inner = 10000
Note : set SSQ to be a large number to avoid an unreasonable number of e1_search
6
maxgap_bad = integer [log 0.05 / log (1 - given_e2)]
Note : this is an approximated maximum gap length in the bad state with BER = given_e2
5

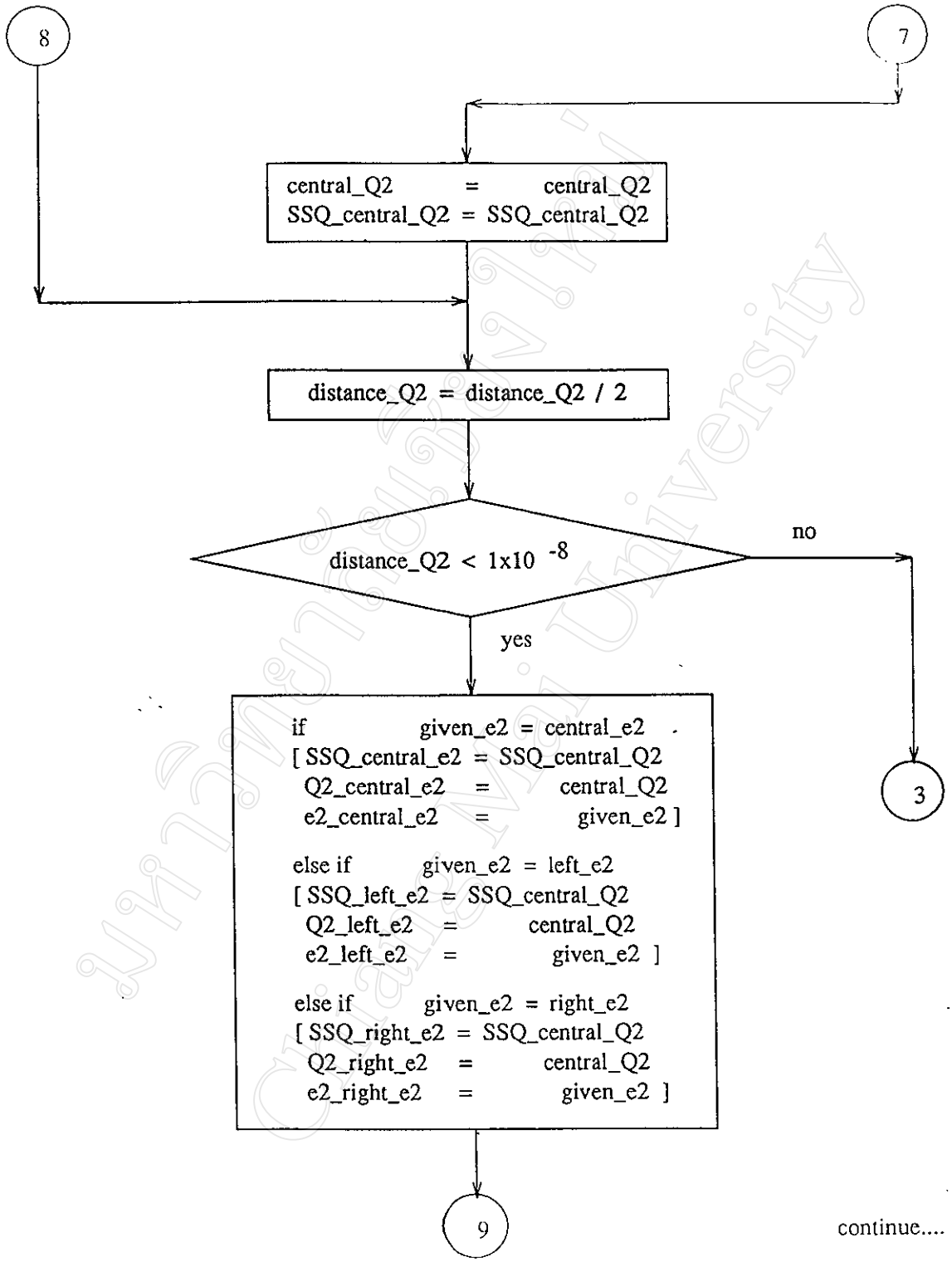
continue....


5
determine : bad_to_good, : the number of changing state from the bad state
bad_to_bad : to the good state and from the bad state to
: itself, giving the condition that if a gap length
: is less than or equal to maxgap_bad, the state
: of the channel is the bad state else the state is
: the good state.
from data file
q21_search = bad_to_good / (bad_to_good + bad_to_bad)
q12_search = given_Q2 * q21_search
(1 - given_Q2)
0 <= q12_search <= 1
no
yes
SSQ_inner = 10000
Note : set SSQ to be a large number to avoid an unreasonable value of q12_search
Q(x)_model = [(1-given_Q2) (1-e1_search) x]
+ [given_Q2*(1-given_e2) x]
SSQ_inner = [Q(x)_data - Q(x)_model]
6

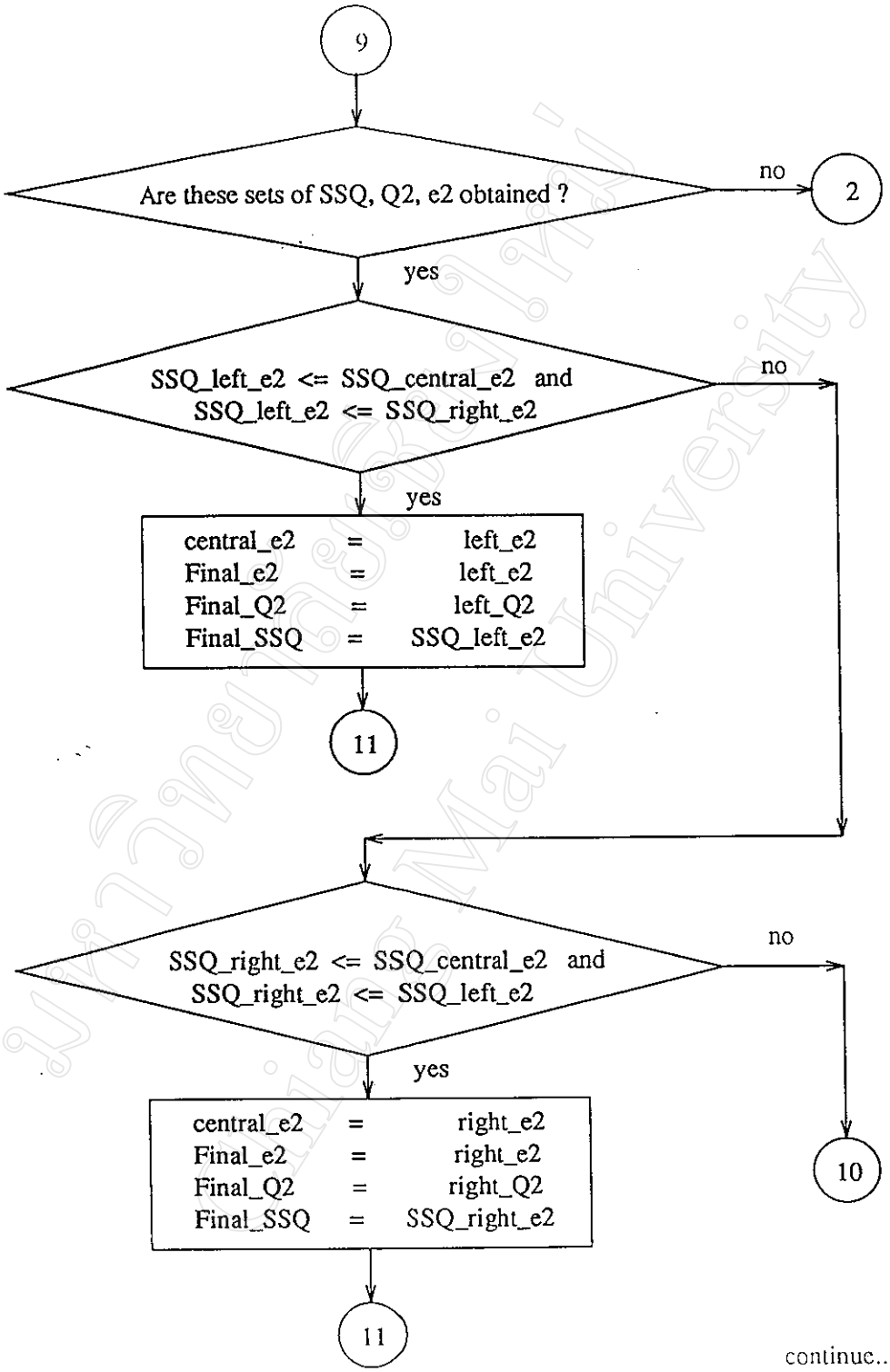
continue....


6
SSQ_central_Q2 = SSQ_inner if given_Q2 = central_Q2
SSQ_left_Q2 = SSQ_inner if given_Q2 = left_Q2
SSQ_right_Q2 = SSQ_inner if given_Q2 = right_Q2
Are these three SSQ obtained ?
no
4
yes
SSQ_left_Q2 <= SSQ_central_Q2 and SSQ_left_Q2 <= SSQ_right_Q2
no
yes
central_Q2 = left_Q2
SSQ_central_Q2 = SSQ_left_Q2
8
SSQ_right_Q2 <= SSQ_central_Q2 and SSQ_right_Q2 <= SSQ_left_Q2
no
yes
central_Q2 = right_Q2
SSQ_central_Q2 = SSQ_right_Q2
7
8

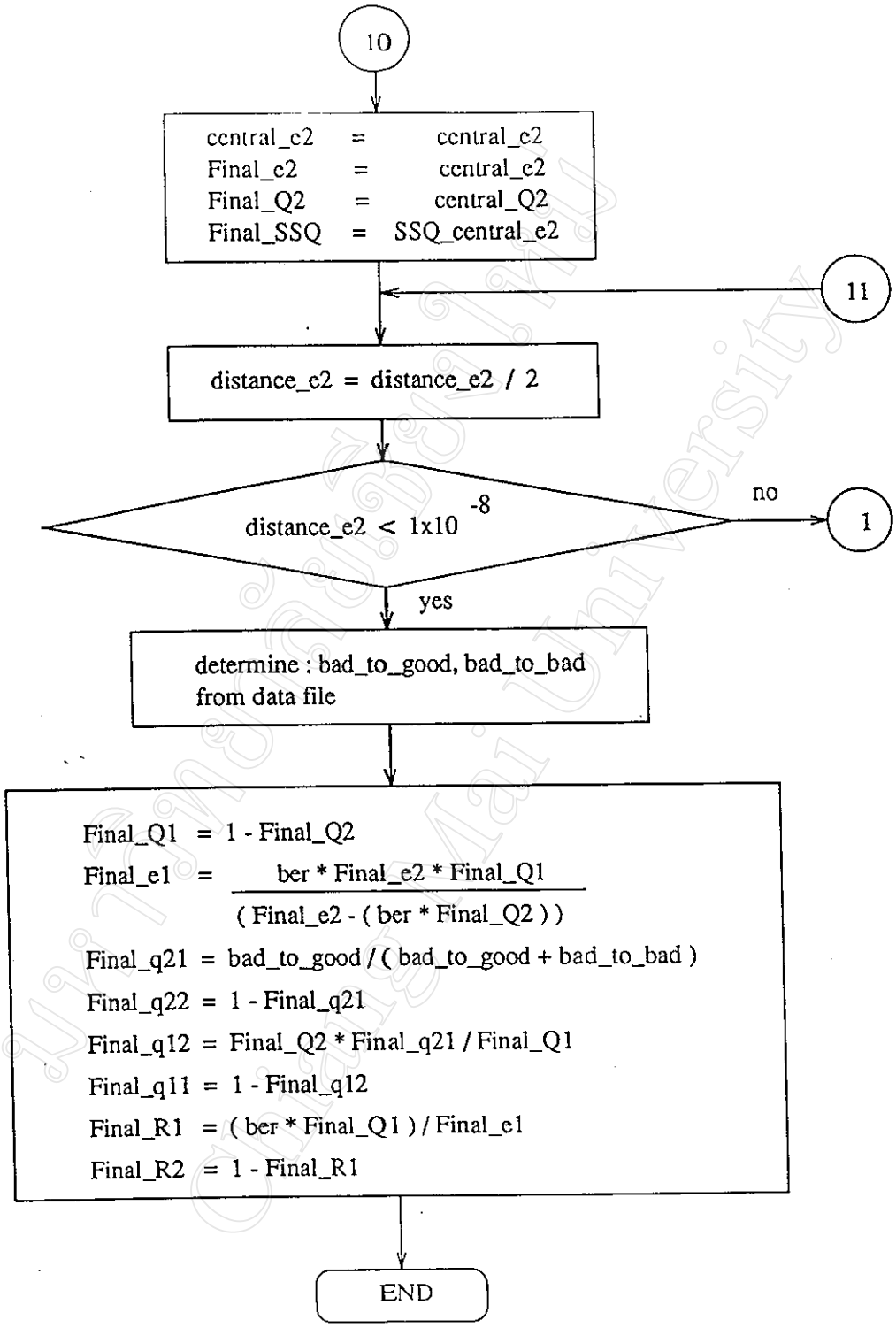
continue....


8
7
central_Q2 = central_Q2
SSQ_central_Q2 = SSQ_central_Q2
distance_Q2 = distance_Q2 / 2
distance_Q2 < 1x10 -8
no
yes
3
if given_e2 = central_e2
[SSQ_central_e2 = SSQ_central_Q2
Q2_central_e2 = central_Q2
e2_central_e2 = given_e2]
else if given_e2 = left_e2
[SSQ_left_e2 = SSQ_central_Q2
Q2_left_e2 = central_Q2
e2_left_e2 = given_e2]
else if given_e2 = right_e2
[SSQ_right_e2 = SSQ_central_Q2
Q2_right_e2 = central_Q2
e2_right_e2 = given_e2]
9

continue....


9
Are these sets of SSQ, Q2, e2 obtained ?
no
2
yes
SSQ_left_e2 <= SSQ_central_e2 and
SSQ_left_e2 <= SSQ_right_e2
no
yes
central_e2 = left_e2
Final_e2 = left_e2
Final_Q2 = left_Q2
Final_SSQ = SSQ_left_e2
11
SSQ_right_e2 <= SSQ_central_e2 and
SSQ_right_e2 <= SSQ_left_e2
no
yes
central_e2 = right_e2
Final_e2 = right_e2
Final_Q2 = right_Q2
Final_SSQ = SSQ_right_e2
10
11

continue...


10
central_e2 = central_e2
Final_e2 = central_e2
Final_Q2 = central_Q2
Final_SSQ = SSQ_central_e2
11
distance_e2 = distance_e2 / 2
distance_e2 < 1x10 -8
no
1
yes
determine : bad_to_good, bad_to_bad
from data file
Final_Q1 = 1 - Final_Q2
Final_e1 = ber * Final_e2 * Final_Q1 / (Final_e2 - (ber * Final_Q2))
Final_q21 = bad_to_good / (bad_to_good + bad_to_bad)
Final_q22 = 1 - Final_q21
Final_q12 = Final_Q2 * Final_q21 / Final_Q1
Final_q11 = 1 - Final_q12
Final_R1 = (ber * Final_Q1) / Final_e1
Final_R2 = 1 - Final_R1
END

ภาคผนวก ข

โปรแกรมภาษา C ที่ใช้ในการจำลองแบบและคำนวณค่าพารามิเตอร์ต่างๆ

มหาวิทยาลัยเชียงใหม่
Chiang Mai University

C Programs for Channel Modelling

1. Program Descriptions and Listings

We provide a functional explanation of the structure of each of the C programs used for channel modelling in association with its listing.

2. Program Structure

We provide a graphical description of the major procedures and their inter–relationships.

ลิขสิทธิ์มหาวิทยาลัยเชียงใหม่
Chiang Mai University

1. Program Descriptions and Listings

All programs for channel modelling are written in C language.

The function of each program is described in association with its listing as follows.

"tsaigen.c"

"interleave.c"

"pmn_anal.c"

"arrg_pmn.c"

"gapconv.c"

"gap_erfr.c"

"burst_int.c"

"meth1_par.c"

"meth2_par.c"

"erfr_cal.c"

"pmn_cal."

"datsmgen.c"

มหาวิทยาลัยเชียงใหม่ Chiang Mai University

"tsaigen.c"

is a program to produce a synthetic data sequence in which '0' represents an error–free bit and '1' represents an errored bit. This program uses a 3–state partitioned Markov chain model, as in Tsai's paper , with parameter values calibrated from the error behaviour of a real HF channel, in order to generate each bit of data. To have a data sequence that long enough to ensure a statistical stability of the error characteristics, we produce an original synthetic data sequence of $4x10^6$ bits. In addition, the program employs a high quality uniform random number generating technique to avoid biases in the {0,1} sequence inherited from the underlying generator.

Note: Before running the program, an output filename is essentially given. In the listing of the program attached, an example of the output filename is "data2.dat".

```
/*-------------------------------------------------------------*
 *                                                             *
 *  Program to generate error seqence by using                 *
 *  3-state Partitioned Markov Chain Model                     *
 *  ( Tsai, IEEE Trans. Comm. vol. COM-17,                     *
 *    no. 1, Feb 1969. )                                       *
 *                                                             *
 *  Date :                                                     *
 *  By  : Adisak  Jaisilp                                      *
 *                                                             *
 *-------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MAXDAT 4000000

FILE *fout;

int next_st();
float rannum();

main()
{  int st, next, datchar;
   long iter;
   float k;

   srand(3);

   fout = fopen("data2.dat","a");

   st = 1;
   iter = 0;
   while ( iter < MAXDAT )
   {  if ( st == 1 )
      {  k =  rannum();
         if ( k >= 0.99911 ) st = 3;
         datchar = '0';
         fputc(datchar, fout);
      }
      else if ( st == 2 )
      {  k =  rannum();
         if ( k >= 0.73644 ) st = 3;
         datchar = '0';
         fputc(datchar, fout);
      }
      else
      {  k = rannum();
         if ( k <= 0.36258 ) st = 1;
         else if ( k >= 0.947768 ) st = 3;
         else
           st = 2;
         datchar = '1';
         fputc(datchar, fout);
      }
```

ลิขสิทธิ์มหาวิทยาลัยเชียงใหม่ Chiang Mai University

```
      ++iter;
   }

   fclose(fout);

}

float rannum()
{  float d1, d2, d3, d4, r;
   int d5;
   d1 = rand()/pow(2.0, 31.0);
   d2 = rand()/pow(2.0, 31.0);
   d3 = rand()/pow(2.0, 31.0);
   d4 = d1 + d2 + d3;
   d5 = d4;
   r = d4 – d5;
   return r;
}
```

มหาวิทยาลัยเชียงใหม่ Chiang Mai University

"interleave.c"

is a program to simulate interleaving an original data sequence of '0's and '1's. The input parameters for the program are packet size (n) and interleaving depth (l). The program reads data n bits from the original sequence and stores in the first row of an array size lxn. The process of reading and storing is continued until the array is filled with l rows of n data bits. The stored data are then read out by columns. During read–out from the array, the order of the data sequence will be rearranged from $1, 2, 3, 4, \ldots\ldots$ to be $1, n+1, n+2, n+3, \ldots\ldots$ and so on. The program continues the process until all of the original data are rearranged. Since the process reads data in and out as block of data lxn bits, the program will discard the last block if its size is not lxn bits. The reason is that we can not simulate the error behaviour since the block contains "blanks" which are not known either '0' nor '1'. This might cause the average BER of the interleaved data to be slightly different from the original one.

Note: Before running the program, two variables and input/output filenames are essentially given. The two variables are **"code_size"** refers to a block length of the interleaver and **"depth"** refers to an interleaving depth.

In the listing attached, an example of the input filename is "data1.dat" and that of the output filename is "n012_l005.dat1".

```
/*-------------------------------------------------------------*
 *                                                             *
 *          Program to simulate the interleaved data sequence  *
 *                                                             *
 *          Date  :                                            *
 *          By    : Adisak Jaisilp                             *
 *                                                             *
 *-------------------------------------------------------------*/

#include <stdio.h>

FILE *fpin;
FILE *fpout;

main()
{  int n, l, dat[390][165], code_size, depth;

   code_size = 12;
   depth = 5;

   fpin = fopen("data1.dat", "r");
   fpout = fopen("n012_l005.dat1", "a");

   n = l = 1;

   while ( (dat[n][l] = fgetc(fpin)) != EOF )
   {  if ( l < depth )
      {  if ( n < code_size ) ++n;
         else
         {  n = 1;
            ++l;
         }
      }
      else if ( l == depth && n < code_size ) ++n;
      else
      {  n = l = 1;
         while ( n <= code_size )
         {  while ( l <= depth )
            {  fputc( dat[n][l], fpout );
               ++l;
            }
            l = 1;
            ++n;
         }
         n = l = 1;
      }
   }

   fclose(fpin);
   fclose(fpout);
}
```

ลิขสิทธิ์มหาวิทยาลัยเชียงใหม่
Chiang Mai University

"pmn_anal.c"

is the program to determine block error statistics $P(m,n)$ of the {0,1} data sequence. The algorithm of the program is as follows. The data sequence is divided into blocks each size n bits and the number of blocks which contain m errors is counted. The probability of $P(m,n)$ is the ratio of the number of blocks containing m errors to the total number of divided blocks. We set the parameters of the program to determine $P(m,n)$ for $m = 0, 1, 2, 3, 4, 5$ and $n = m, m+1, m+2, ..., 30$.

Note: Before running the program, one variable and two input/output filenames are essentially given. The variable is **"ber"** refers to the average bit error rate of the data sequence. In the listing attached, an example of the input filename is **"dsto_sim1.dat"** and that of the output filename is **"dsto_sim1.pmn"**.

มหาวิทยาลัยเชียงใหม่ Chiang Mai University

```
/*-------------------------------------------------------------*
 *                                                             *
 *          Program to determine the probability of m errors in *
 *           block of data length n.....P ( m, n )             *
 *                                                             *
 *          Date  :                                            *
 *          By    : Adisak  Jaisilp                            *
 *                                                             *
 *-------------------------------------------------------------*/

#include <stdio.h>

FILE *fpin;
FILE *fpout;

main()
{  int n, m, max_err, nof_err, i, j, c;
   long tot_blk, blk[30];
   float ber, q, pmn[30], acc_pmn;

   ber = 0.031158;
   q = 1.0 – ber;

   fpout = fopen("dsto_sim1.pmn", "a");

   fprintf(fpout, "#The probability of m errors in data block length n \n\n");
   fprintf(fpout, "#n    m       P(m, n)      accumulative P(m, n) \n\n");

   n = 1;
   m = 0;
   pmn[0] = 1.0 – ber;
   pmn[1] = ber;
   acc_pmn = 0;

   while ( m < 2 )
   {  acc_pmn = acc_pmn + pmn[m];
      fprintf(fpout, "%2u   %2u      %10.9f      %10.9f \n", n, m, pmn[m], acc_pmn );
      ++m;
   }
   n = 2;
   while ( n < 31 )
   {  fpin = fopen("dsto_sim1.dat", "r");
      max_err = nof_err = 0;
      tot_blk = 0;
      i = n;
      j = 0;
      while ( j <= i )
      {  blk[j] = 0;
         ++j;
      }

      i = 0;
      while (( c = fgetc(fpin) ) != EOF )
      {  if ( c == '1' || c == '0' )
         {  if ( i < n )
            {  if ( c == '1' ) ++nof_err;
```

```
            ++i;
          }
          else
          {   if ( max_err < nof_err ) max_err = nof_err;
            ++blk[nof_err];
            ++tot_blk;
            i = nof_err = 0;
          }
        }
      }
      fclose(fpin);

      i = 0;
      acc_pmn = 0;

      while ( i <= max_err )
      {   pmn[i] = (float)blk[i] / (float)tot_blk;
        acc_pmn = acc_pmn + pmn[i];
        fprintf(fpout, "%2u    %2u        %10.9f      %10.9f\n", n, i, pmn[i], acc_pmn );
        ++i;
      }
      ++n;
    }

    fclose(fpout);

}
```

มหาวิทยาลัยเชียงใหม่ Chiang Mai University

"arrg_pmn.c"

is the program to rearrange the output file from "pmn_anal.c" in which the $P(m,n)$ are in the mixed form ($m = 0, 1, 2, 3, 4, 5$ and $n = m, m+1, m+2, ..., 30$). The output file of the "arrg_pmn.c" is a $P(m,n)$ with a specific value of m for example $P(2,n)$.

Note: Before running the program, two input/output filenames are essentially given. In the listing attached, an example of the input filename is "dsto_sim1.pmn" and that of the output filename is "dsto_sim1.p0n". In addition, four points in the program have to be set according to the value of m. Examples of these points corresponding to $m = 0$ are the '0s' in line numbers 37, 41, 42 and 43 of the listing respectively.

มหาวิทยาลัยเชียงใหม่ Chiang Mai University

```
/*-------------------------------------------------------*
 *                                                       *
 *    Program to rearrange the P(m,n)                    *
 *              for plotting                             *
 *                                                       *
 *    Date :                                             *
 *    By   : Adisak  Jaisilp                             *
 *                                                       *
 *-------------------------------------------------------*/

#include <stdio.h>

#define MAXLINE 100

FILE *fpin;
FILE *fpout;

main()
{  int i, n, m;
   float pmn, acc_pmn, p_m_n[35];
   char line[MAXLINE], *lp;

   fpin = fopen("dsto_sim1.pmn", "r");
   i = 0;
   while ( i < 4 )
   {  if (( lp = fgets(line, MAXLINE, fpin) ) != '\0' ) ++i;
   }

   n = 1;
   while ( n <= 30 )
   {  p_m_n[n] = 0;
      ++n;
   }

   while ( ( lp = fgets(line, MAXLINE, fpin ) ) != '\0' )
   {  sscanf(line, "%d %d %f %f", &n, &m, &pmn, &acc_pmn );
      if ( m == 0 ) p_m_n[n] = pmn;
   }
   fclose(fpin);

   fpout = fopen("dsto_sim1.p0n", "a");
   fprintf(fpout, "# P( 0, n ) \n\n");
   fprintf(fpout, "# n         P( 0, n ) \n\n");
   n = 1;
   while ( n <= 30 )
   {  fprintf(fpout, "%u        %10.9f \n", n, p_m_n[n]);
      ++n;
   }
   fclose(fpout);
}
```

"gapconv.c"

is the program to convert the {0,1} data sequence to another form more convenient for the determination of other error statistics and model parameters. The new form of data is as follows:

0 674

1 1

0 53

1 2

0 192

1 1 ..... and so on.

This is a compressed form of the original data employing "run length encoding". The first column indicates whether the data bit is '0' or '1' and the second column is the number of repetitions before a change occurs.

Note: Before running the program, two input/output filenames are essentially given. In the listing attached, an example of the input filename is **"dsto_sim1.dat"** and that of the output filename is **"dstosim1_gap.dat"**.

ลิขสิทธิ์มหาวิทยาลัยเชียงใหม่
Chiang Mai University

```
/*-------------------------------------------------------------*
 *                                                             *
 *   Program to convert a sequence of 0 and 1 characters       *
 *      into a list of 2 columns which indicates               *
 *      the amount of consecutive data                         *
 *                                                             *
 *   Date :                                                    *
 *   By  : Adisak Jaisilp                                      *
 *                                                             *
 *-------------------------------------------------------------*/

#include <stdio.h>

FILE *fpin;
FILE *fpout;

main()
{  int dat, zero_acc, one_acc, last_dat;

   fpin = fopen("dsto_sim1.dat", "r");
   fpout = fopen("dstosim1_gap.dat", "a");

   zero_acc = 0;
   one_acc = 0;
   last_dat = fgetc(fpin);
   if ( last_dat == '0' ) ++zero_acc;
   else if ( last_dat == '1' ) ++one_acc;
   while ( ( dat = fgetc(fpin) ) != EOF )
   {  if ( dat == '0' || dat == '1' )
      {  if ( dat == '0' )
         {  ++zero_acc;
            if ( last_dat == '1' )
            {  fprintf(fpout, "1 % 10u \n", one_acc);
               one_acc = 0;
            }
            last_dat = dat;
         }
         else
         {  ++one_acc;
            if ( last_dat == '0' )
            {  fprintf(fpout, "0 % 10u \n", zero_acc);
               zero_acc = 0;
            }
            last_dat = dat;
         }
      }
   }
   if ( zero_acc == 0 )
   {  fprintf(fpout, "1 % 10u \n", one_acc);
      one_acc = 0;
   }

   else if ( one_acc == 0 )
   {  fprintf(fpout, "0 % 10u \n", zero_acc);
      zero_acc = 0;
   }
```

ลิขสิทธิ์มหาวิทยาลัยเชียงใหม่
Chiang Mai University

```
    fclose(fpin);
    fclose(fpout);
}
```

ลิขสิทธิ์มหาวิทยาลัยเชียงใหม่
Chiang Mai University

"gap_erfr.c"

is the program to determine the average bit error rate, the gap distribution and the error–free run distribution of the data. The input data is the compressed form of the original data provided by "**gapconv.c**" described above.

Note: Before running the program, three input/output filenames are essentially given. In the listing attached, an example of the input filename is "**dstosim1_gap.dat**" and those of the output filenames are "**dsto_sim1.gap**" refers to the gap distribution data file and "**dsto_sim1.erf**" refers to the error–free run distribution data file.

มหาวิทยาลัยเชียงใหม่ Chiang Mai University

```
/*-------------------------------------------------------------------*
 *                                                                   *
 *   Program to determine the gap distribution and the error-free run *
 *      distribution which are ready for plotting graphs             *
 *                                                                   *
 *   Date :                                                          *
 *   By   : Adisak Jaisilp                                           *
 *                                                                   *
 *-------------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>

FILE *fpin;
FILE *fpgap;
FILE *fperf;

main()
{  int fr_arg, bk_arg, gap_len[10][5], i, j, k;
   int sig_nal;
   long int totdata, amt, totgaps, toterror, acc_gaps, gaps, x, erfr_len;
   float ber, acc_prob[10][5], y, erfr_run[10][5], erfrbase, z;

   fpin = fopen("dstosim1_gap.dat", "r");
   fpgap = fopen("dsto_sim1.gap", "a");
   fperf = fopen("dsto_sim1.erf", "a");

   fprintf(fpgap, "#The Gap Distribution\n\n");

   totdata = totgaps = toterror = 0;
   for ( j = 0 ; j <= 4 ; ++j )
   {  for ( i = 1 ; i <= 9 ; ++i )
      {  gap_len[i][j] = acc_prob[i][j] = 0;
      }
   }
   while ( fscanf(fpin, "%u %u ", &sig_nal, &amt ) != EOF )
   {  totdata = totdata + amt;
      if ( sig_nal == 1 )
        toterror = toterror + amt;
      else
      {  if ( amt < 10 )
         {  fr_arg = amt;
            bk_arg = 0;
         }
         else if ( amt >= 10 && amt <= 90 )
         {  z = ( amt + 9.0 )/10.0;
            fr_arg = z;
            bk_arg = 1;
         }
         else if ( amt > 90 && amt <= 900 )
         {  z = ( amt + 99.0 )/100.0;
            fr_arg = z;
            bk_arg = 2;
         }
         else if ( amt > 900 && amt <= 9000 )
         {  z = ( amt + 999.0 )/1000.0;
```

```
            fr_arg = z;
            bk_arg = 3;
        }
        else if ( amt > 9000 && amt <= 90000 )
        {   z = ( amt + 9999.0 )/10000.0;
            fr_arg = z;
            bk_arg = 4;
        }
        else
        {   fr_arg = 1;
            bk_arg = 5;
        }
        ++gap_len[fr_arg][bk_arg];
        totgaps = totgaps + 1;
    }
}

ber = (float)toterror/(float)totdata;
fprintf(fpgap, "#The average bit error rate ( BER ) = %10.8f\n\n", ber);
fprintf(fpgap, "#Gap length      Cumu. rel. freq.\n\n");
j = 0;
i = 1;
acc_gaps = 0;

while ( j < 5 )
{   while ( i < 10 )
    {   acc_gaps = acc_gaps + (long)gap_len[i][j];
        acc_prob[i][j] = (float)acc_gaps / (float)totgaps;
        gaps = (long)i * (long)pow(10.0,(float)(j+1)) / (long)10;
        fprintf(fpgap, "%8u %20.6f\n", gaps, acc_prob[i][j]);
        ++i;
    }
    i = 1;
    ++j;
}
x = 100000;
y = 1.000000;
fprintf(fpgap, "%8u %20.6f\n\n", x, y);
fclose(fpgap);

erfrbase = (float)totgaps / (float)toterror;
fprintf(fperf, "#The Error-free Run Distribution\n\n");
fprintf(fperf, "#Error-free run length     Cumu. rel. freq.\n\n");
j = 0;
i = 1;
acc_prob[0][0] = 0;
while ( j < 5 )
{   while ( i < 10 )
    {   if ( j == 0 ) k = i - 1;
        else k = i;
        erfr_run[i][j] = ( 1.0 - acc_prob[k][j] ) * erfrbase;
        erfr_len = (long)i * (long)pow(10.0,(float)(j+1)) / (long)10;
        fprintf(fperf, "%15u %20.6f\n", erfr_len, erfr_run[i][j]);
        ++i;
    }
```

```
        i = 1;
        ++j;
    }
    erfr_run[1][5] = 0;
    erfr_len = 100000;
    fprintf(fperf, "%15u %20.6f\n", erfr_len, erfr_run[1][5]);

    fclose(fperf);

    fclose(fpin);

}
```

มหาวิทยาลัยเชียงใหม่ Chiang Mai University

"burst_int.c"

is the program to determine the burst and burst interval distributions of the data for the error density (Δ_o) currently equal to 0.1. However, the value of parameter Δ_o can be reset inside the program. The input data is also the compressed form of the original data.

Note: Before running the program, three input/output filenames are essentially given. In the listing attached, an example of the input filename is **"dstosim1_gap.dat"** and those of the output filenames are **"dsto_sim1.brst"** refers to the burst distribution dada file and **"dsto_sim1.intv"** refers to the burst interval distribution data file.

มหาวิทยาลัยเชียงใหม่ Chiang Mai University

```
/*-------------------------------------------------------------*
*                                                             *
*    Program to determine the burst distribution              *
*                and the burst interval distribution          *
*                                                             *
*    Date :                                                   *
*    By   : Adisak  Jaisilp                                   *
*                                                             *
*-------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>

FILE *fpin;
FILE *fpout;
FILE *fpintv;

main()
{  int sig_nal, tot_err, fr_arg, bk_arg, round, i, j, k, m, n, bi;
   long r, tot_bit, burst, brst_ln[10][4], tot_brst, amt;
   long tot_intv, interval, intv, interv_ln[10][6];
   float denst, brst_dis, brst_pro, intv_pro, intv_dis;

   fpin = fopen("dstosim1_gap.dat", "r");
   fpout = fopen("dsto_sim1.brst", "a");
   fpintv = fopen("dsto_sim1.intv", "a");
   fprintf(fpout, "# The burst distribution of DSTO_SIM1.DAT \n");
   fprintf(fpintv, "# The burst interval distribution of DSTO_SIM1.DAT \n");

   tot_err = 0;
   tot_brst = r = tot_bit = burst = tot_intv = 0;
   n = bi = 0;

   for ( i = 1 ; i <= 6 ; ++i )
   {  for ( j = 1 ; j <= 9 ; ++j )
      {  brst_ln[j][i] = interv_ln[j][i] = 0;
      }
   }
   while ( fscanf(fpin, "%u %u \n", &sig_nal, &amt ) != EOF )
   {  if ( bi > 0 ) intv = intv + amt;
      if ( sig_nal == 1 )
      {  ++r;
         tot_bit = tot_bit + amt;
         tot_err = tot_err + amt;
         denst = (float)tot_err / (float)tot_bit;
         if ( denst < 0.1 )
         {  if ( r > (long)2 )
            {  if ( burst < (long)10 )
               {  fr_arg = (int)burst;
                  bk_arg = 1;
               }
               else if ( burst >= (long)10 && burst <= (long)90 )
               {  fr_arg = (int)( ( burst + (long)9 )/(long)10 );
                  bk_arg = 2;
               }
               else if ( burst > (long)90 && burst <= (long)900 )
```

```
    {   fr_arg = (int)( ( burst + (long)99 )/(long)100 );
        bk_arg = 3;
    }
    else
    {   fr_arg = 1;
        bk_arg = 4;
    }
    ++brst_ln[fr_arg][bk_arg];
    ++tot_brst;
    intv = tot_bit – burst;
    ++bi;
  }
  tot_bit = amt;
  tot_err = amt;
  r = 1;
}
else
{   if ( r > (long)1 )
    {   burst = tot_bit;
        if ( r = (long)2 )
        {   if ( bi > 0 )
            {   interval = intv – tot_bit;
                bi = 0;
                if ( interval < (long)10 )
                {   fr_arg = (int)interval;
                    bk_arg = 1;
                }
                else if ( interval >= (long)10 && interval <= (long)90 )
                {   fr_arg = (int)( ( interval + (long)9 )/(long)10 );
                    bk_arg = 2;
                }
                else if ( interval > (long)90 && interval <= (long)900 )
                {   fr_arg = (int)( ( interval + (long)99 )/(long)100 );
                    bk_arg = 3;
                }
                else if ( interval>(long)900 && interval<=(long)9000 )
                {   fr_arg = (int)((interval+(long)999)/(long)1000);
                    bk_arg = 4;
                }
                else if ( interval>(long)9000 && interval<=(long)90000 )
                {   fr_arg = (int)((interval+(long)9999)/(long)10000);
                    bk_arg = 5;
                }
                else
                {   fr_arg = 1;
                    bk_arg = 6;
                }
                ++interv_ln[fr_arg][bk_arg];
                ++tot_intv;
            }
        }

    }
}
```

```
    }
    else
    {   if ( r >= (long)l ) tot_bit = tot_bit + amt;
    }
}

fclose(fpin);

m = k = 0;
brst_dis = 0;
fprintf(fpout, "# Burst density >= 0.1 \n\n");
fprintf(fpout, "# Burst length     Cumu. rel. frequency \n\n");

for ( i = 1 ; i <= 3 ; ++i )
{   for ( j = 1 ; j <= 9 ; ++j )
    {   ++m;
        k = (int)( (float)j * pow(10.0, (float)i)/10.0 );
        brst_pro = (float)brst_ln[j][i] / (float)tot_brst;
        brst_dis = brst_dis + brst_pro;
        fprintf(fpout, "%8u          %10.9f \n", k, brst_dis);

    }
}

brst_dis = brst_dis + ( (float)brst_ln[1][4] / (float)tot_brst );
k = 1000;
fprintf(fpout, "%8u          %10.9f \n", k, brst_dis);
fclose(fpout);

intv_dis = 0;

fprintf(fpintv, "# Burst density >= 0.1 \n\n");
fprintf(fpintv, "# Burst interval length     Cumu. rel. frequency \n\n");

for ( i = 1 ; i <= 5 ; ++i )
{   for ( j = 1 ; j <= 9 ; ++j )
    {   k = (int)( (float)j * pow(10.0, (float)i)/10.0 );
        intv_pro = (float)interv_ln[j][i] / (float)tot_intv;
        intv_dis = intv_dis + intv_pro;
        fprintf(fpintv, "%8u                  %10.9f \n", k, intv_dis);
    }
}

intv_dis = intv_dis + ( (float)interv_ln[1][6] / (float)tot_intv );
k = 100000;
fprintf(fpout, "%8u                  %10.9f \n", k, intv_dis);
fclose(fpintv);

}
```

"meth1_par.c"

is the program to determine the model parameters by means of the optimisation algorithm which uses the state separator, as described in Appendix This program requires two input data files: the error–free run distribution and the compressed form of the original data.

Note: Before running the program, three input/output filenames are essentially given. In the listing attached, examples of the input filenames are **"dsto_gap.dat"** refers to the reformatted data file and **"dsto.erf"** refers to the error–free run distribution file and that of the output filename is **"dsto_m1.par"**. There are three points in the program needed to fill in with the filename **"dsto_gap.dat"**.

มหาวิทยาลัยเชียงใหม่ Chiang Mai University

```
/*-------------------------------------------------------------*
*                                                             *
*   Program to determine the parameters of 2-state Markov chain model
*       by using the optimization technique associated with
*         state seperator
*
*   Date  :
*   By    : Adisak  Jaisilp
*
*-------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>

#define MAXLINE 100

FILE *fpin;
FILE *fperf;
FILE *fpout;

main()
{  int i, c, j, m;
   int sig_nal, cur_st, next_st, fcur_st, bcur_st, fnext_st, bnext_st;
   long erfr_len, max_gaps, amt, totdata, toterror, serspace;
   long cent_sep, gap_st1, gap_st2, nof_gaps, cent_q11, cent_q12, cent_q22;
   long cent_q21, distance, frnt_sep, back_sep, fgap_st1, fgap_st2;
   long bgap_st1, bgap_st2;
   long frnt_q11, frnt_q22, frnt_q12, frnt_q21, back_q11, back_q22, back_q21;
   long back_q12;

   float erfr[47], ber, ber_st1, ber_st2, k, cq11, cq22, cq21, cq12;
   float q1c, q2c, cqx, censq_er, fsq_er, bsq_er, fber_st1, fber_st2;
   float fq11, fq12, fq22, fq21, q1f, q2f, fqx, bqx, bber_st1, bber_st2;
   float bq11, bq12, bq22, bq21, q1b, q2b, p1, p2;
   float q_st1, q_st2, lsq_err, fq_st1, fq_st2, bq_st1, bq_st2;
   float flsq_err, blsq_err, ber_cal;

   char line[MAXLINE], *lp;

   fperf = fopen("dsto.erf", "r");
   i = 0;
   while ( i < 4 )
   {  if ( (lp = fgets(line, MAXLINE, fperf)) != '\0' ) ++i;
   }
   i = 1;
   while ( i < 47 )
   {  if ( (lp = fgets(line, MAXLINE, fperf)) != '\0' )
      {  sscanf(line, "%d %f", &erfr_len, &erfr[i]);
      }
      ++i;
   }
   fclose(fperf);

   fpin = fopen("dsto_gap.dat", "r");
   max_gaps = 0;
   totdata = toterror = 0;
```

```
while ( fscanf(fpin, "%u %u", &sig_nal, &amt) != EOF )
{   totdata = totdata + amt;
   if ( sig_nal == 1 ) toterror = toterror + amt;
   if ( sig_nal == 0 && amt > max_gaps )
      max_gaps = amt;
}
fclose(fpin);

ber = (float)toterror / (float)totdata;
cent_sep = max_gaps / (long)2;
fpin = fopen("dsto_gap.dat", "r");
cur_st = 1;
nof_gaps = 0;
gap_st1 = gap_st2 = 0;
while ( fscanf(fpin, "%u %u", &sig_nal, &amt) != EOF )
{  if ( sig_nal == 0 )
   {  if ( amt > cent_sep )
      {  next_st = 1;
         gap_st1 = gap_st1 + amt;
         if ( cur_st == 1 ) ++cent_q11;
         if ( cur_st == 2 ) ++cent_q21;
      }
      else
      {  next_st = 2;
         gap_st2 = gap_st2 + amt;
         if ( cur_st == 1 ) ++cent_q12;
         if ( cur_st == 2 ) ++cent_q22;
      }
      ++nof_gaps;
   }
   else
   {  if ( amt > 1 )
      {  next_st = 2;
         cent_q22 = cent_q22 + amt - (long)1;
         nof_gaps = nof_gaps + amt - (long)1;
      }
   }
   cur_st = next_st;
}
fclose(fpin);

ber_st1 = 1.0 / ( 1.0 + ( (float)gap_st1 / (float)nof_gaps ) );
ber_st2 = 1.0 / ( 1.0 + ( (float)gap_st2 / (float)nof_gaps ) );
cq11 = (float)cent_q11 / ( (float)cent_q11 + (float)cent_q12 );
cq12 = 1.0 - cq11;
cq22 = (float)cent_q22 / ( (float)cent_q22 + (float)cent_q21 );
cq21 = 1.0 - cq22;
q1c = cq21 / ( 1.0 - cq11 + cq21 );
q2c = 1.0 - q1c;

i = j = m = 1;
censq_er = 0;
while ( i < 6 )
{  while ( j < 10 )
```

```
    {   k = (float)j * pow(10.0,(float)i) / 10.0;
        q_st1 = 1.0 – ber_st1;
        q_st2 = 1.0 – ber_st2;
        cqx = ( q1c * pow(q_st1,k) ) + ( q2c * pow(q_st2,k) );

        lsq_err = erfr[m] - cqx;
        censq_er = censq_er + pow(lsq_err,2.0);
        ++m;
        ++j;
    }
    j = 1;
    ++i;
}

serspace = max_gaps;
distance = serspace / (long)4;

fpout = fopen("dsto_m1.par", "a");
fprintf(fpout, "The actual Bit Error Rate ( BER ) = %10.9f\n\n", ber);
fprintf(fpout, "The state separator      Summation of Square Error\n\n");
fprintf(fpout, "%10u                    %9.8f\n", cent_sep, censq_er);

while ( serspace > 1 )
{   frnt_sep = cent_sep + distance;
    back_sep = cent_sep – distance;

    fpin = fopen("dsto_gap.dat", "r");
    nof_gaps = 0;
    fcur_st = bcur_st = 1;
    fgap_st1 = fgap_st2 = bgap_st1 = bgap_st2 = 0;
    frnt_q11 = frnt_q21 = frnt_q12 = frnt_q22 = 0;
    back_q11 = back_q12 = back_q21 = back_q22 = 0;
    fsq_er = bsq_er = 0;

    while ( fscanf(fpin, "%u %u", &sig_nal, &amt) != EOF )
    {   if ( sig_nal == 0 )
        {   if ( amt > frnt_sep )
            {   fnext_st = 1;
                fgap_st1 = fgap_st1 + amt;
                if ( fcur_st == 1 ) ++frnt_q11;
                if ( fcur_st == 2 ) ++frnt_q21;
            }
            else
            {   fnext_st = 2;
                fgap_st2 = fgap_st2 + amt;
                if ( fcur_st == 1 ) ++frnt_q12;
                if ( fcur_st == 2 ) ++frnt_q22;
            }
            if ( amt > back_sep )
            {   bnext_st = 1;
                bgap_st1 = bgap_st1 + amt;
                if ( bcur_st == 1 ) ++back_q11;
                if ( bcur_st == 2 ) ++back_q21;
            }
            else
```

```
      {  bnext_st = 2;
         bgap_st2 = bgap_st2 + amt;
         if ( bcur_st == 1 ) ++back_q12;
         if ( bcur_st == 2 ) ++back_q22;
      }
      ++nof_gaps;
    }
    else
    {  if ( amt > 1 )
       {  fnext_st = bnext_st = 2;
          frnt_q22 = frnt_q22 + amt - (long)1;
          back_q22 = back_q22 + amt - (long)1;
          nof_gaps = nof_gaps + amt - (long)1;
       }
    }
    fcur_st = fnext_st;
    bcur_st = bnext_st;
}
fclose(fpin);

fber_st1 = 1.0 / ( 1.0 + ( (float)fgap_st1 / (float)nof_gaps ) );
fber_st2 = 1.0 / ( 1.0 + ( (float)fgap_st2 / (float)nof_gaps ) );
fq11 = (float)frnt_q11 / ( (float)frnt_q11 + (float)frnt_q12 );
fq12 = 1.0 - fq11;
fq22 = (float)frnt_q22 / ( (float)frnt_q22 + (float)frnt_q21 );
fq21 = 1.0 - fq22;
q1f = fq21 / ( 1.0 - fq11 + fq21 );
q2f = 1.0 - q1f;

bber_st1 = 1.0 / ( 1.0 + ( (float)bgap_st1 / (float)nof_gaps ) );
bber_st2 = 1.0 / ( 1.0 + ( (float)bgap_st2 / (float)nof_gaps ) );
bq11 = (float)back_q11 / ( (float)back_q11 + (float)back_q12 );
bq12 = 1.0 - bq11;
bq22 = (float)back_q22 / ( (float)back_q22 + (float)back_q21 );
bq21 = 1.0 - bq22;
q1b = bq21 / ( 1.0 - bq11 + bq21 );
q2b = 1.0 - q1b;

i = j = m = 1;
while ( i < 6 )
{  while ( j < 10 )
   {  k = (float)j * pow(10.0,(float)i) / 10.0;
      fq_st1 = 1.0 - fber_st1;
      fq_st2 = 1.0 - fber_st2;
      bq_st1 = 1.0 - bber_st1;
      bq_st2 = 1.0 - bber_st2;
      fqx = ( q1f * pow(fq_st1,k) )+( q2f * pow(fq_st2,k) );
      bqx = ( q1b * pow(bq_st1,k) )+( q2b * pow(bq_st2,k) );
      flsq_err = erfr[m] - fqx;
      blsq_err = erfr[m] - bqx;
      fsq_er = fsq_er + pow(flsq_err,2.0);
      bsq_er = bsq_er + pow(blsq_err,2.0);
      ++m;
      ++j;
```

```
    }
    j = 1;
    ++i;
  }

  if ( fsq_er < censq_er && fsq_er < bsq_er )
  {  ber_st1 = fber_st1;
     ber_st2 = fber_st2;
     cq11 = fq11;
     cq12 = fq12;
     cq22 = fq22;
     cq21 = fq21;
     q1c = q1f;
     q2c = q2f;
     censq_er = fsq_er;
     cent_sep = frnt_sep;
  }
  if ( bsq_er < censq_er && bsq_er < fsq_er )
  {  ber_st1 = bber_st1;
     ber_st2 = bber_st2;
     cq11 = bq11;
     cq12 = bq12;
     cq22 = bq22;
     cq21 = bq21;
     q1c = q1b;
     q2c = q2b;
     censq_er = bsq_er;
     cent_sep = back_sep;
   }

   serspace = serspace / (long)2;
   distance = serspace / (long)4;

   fprintf(fpout, "%10u                %9.8f \n", cent_sep, censq_er);
}

ber_cal = 1.0 / ( (q1c/ber_st1) + (q2c/ber_st2) );

fprintf(fpout, "\n\nThe parameters of 2-state Markov chain model are as follow :\n\n");
fprintf(fpout, "The State Separator = %10u\n", cent_sep);
fprintf(fpout, "The Summation of Square Error = %9.8f\n", censq_er);
fprintf(fpout, "The average Bit Error Rate ( calculated BER ) = %10.9f\n", ber_cal);
fprintf(fpout, "The BER in good state ( e1 ) = %10.9f\n", ber_st1);
fprintf(fpout, "The BER in bad state  ( e2 ) = %10.9f\n", ber_st2);
fprintf(fpout, "The transition probability from good state to itself ( q11 ) = %10.9f\n", cq11);
fprintf(fpout, "The transition probability fron bad state to itself ( q22 ) = %10.9f\n", cq22);
fprintf(fpout, "The transition probability from good to bad state ( q12 ) = %10.9f\n", cq12);
fprintf(fpout, "The transition probability from bad to good state ( q21 ) = %10.9f\n", cq21);
fprintf(fpout, "The probability of being in good state after error ( Q1 ) = %10.9f\n", q1c);
fprintf(fpout, "The probability of being in bad state after error  ( Q2 ) = %10.9f\n", q2c);

p1 = ber_cal * q1c / ber_st1;
p2 = ber_cal * q2c / ber_st2;
```

```
    fprintf(fpout, "The unconditional probability of being in good state ( R1 ) = %10.9f\n", p1);
    fprintf(fpout, "The unconditional probability of being in bad state  ( R2 ) = %10.9f\n", p2);


    fclose(fpout);
}
```

มหาวิทยาลัยเชียงใหม่
Chiang Mai University

"meth2_par.c"

is the program to determine the model parameters by means of the optimisation algorithm which uses the bad state parameters, as explained in Appendix . This program also requires two input data files as in the case for **"meth1_par.c"**.

Note: Before running the program, one variable and three input/output filenames are essentially given. The variable is **"ber"** refers to the average bit error rate of the data sequence. In the listing attached, examples of the input filenames are **"dsto.erf"** refers to the error-free run distribution file and **"dsto_gap.dat"** refers to the reformatted data file and that of the output filename is **"real.par"**. There are four points in the program needed to fill in with the filename **"dsto_gap.dat"**.

มหาวิทยาลัยเชียงใหม่ Chiang Mai University

```
/*-------------------------------------------------------------*
 *                                                             *
 *    Program to determine the parameters of 2-state Markov    *
 *        chain model by using optimization method             *
 *        associated with binary search technique              *
 *                                                             *
 *    Date  :                                                  *
 *    By    : Adisak  Jaisilp                                  *
 *                                                             *
 *-------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>

#define MAXLINE 100

FILE *fpin;
FILE *fperf;
FILE *fpout;

main()
{  char line[MAXLINE], *lp;
   int i, j, k, l;
   int sig_nal, cur_st, next_st;
   long erfr_len, amt, mx_gpst2, gap_q21, gap_q22;
   float erfr[47], ber, x, q1, frst_ssq, q2, e1, e2;
   float tot_ssq, q21, q22, q12, q11, ber_cal, p1, p2;
   float cpt, lpt, rpt, distance, erf_cpt, erf_lpt;
   float erf_rpt, ssq_cpt, ssq_lpt, ssq_rpt;
   float e2_ser, e1_cpt, e1_lpt, e1_rpt;
   float center, left, right, dist_out, cent_ssq, cent_q2;
   float cent_e2, left_ssq, left_q2, left_e2, right_ssq;
   float right_q2, right_e2, q21_ser, q12_ser;

   ber = 0.02088542;
   fpout = fopen("real.par", "a");
   fperf = fopen("dsto.erf", "r");

   i = 0;
   while ( i < 4 )
   {  if ( ( lp = fgets(line,MAXLINE,fperf) ) != '\0' ) ++i;
   }
   i = 1;
   while ( i < 47 )
   {  if ( ( lp = fgets(line,MAXLINE,fperf) ) != '\0' )
      {  sscanf(line, "%d %f", &erfr_len, &erfr[i]);
      }
      ++i;
   }
   fclose(fperf);

   tot_ssq = 10000.0;

   center = 0.25;
   dist_out = 0.125;
   while ( dist_out >= 0.00000001 )
```

```
{  left = center – dist_out;
   right = center + dist_out;
   for ( i = 1 ; i < 4 ; i++ )
   {  if ( i == 1 ) e2_ser = center;
      else if ( i == 2 ) e2_ser = left;
      else e2_ser = right;

      cpt = 0.5;
      distance = 0.25;
      e1_cpt = (ber*e2_ser)*(1.0–cpt)/((e2_ser–(ber*cpt)));
      if ( e1_cpt > 0 )
      {  x = ( log(0.05) / log(1.0–e2_ser) ) + 0.5;
         mx_gpst2 = (long)x;
         gap_q21 = gap_q22 = 0;

         fpin = fopen("dsto_gap.dat", "r");

         fscanf(fpin, "%u %u", &sig_nal, &amt);
         if ( sig_nal == 0 )
         {  if ( amt > mx_gpst2 ) cur_st = 1;
            else cur_st = 2;
         }
         while ( fscanf(fpin, "%u %u", &sig_nal, &amt) != EOF )
         {  if ( sig_nal == 0 )
            {  if ( amt > mx_gpst2 )
               {  next_st =1;
                  if ( cur_st == 2 ) ++gap_q21;
               }
               else
               {  next_st = 2;
                  if ( cur_st == 2 ) ++gap_q22;
               }
            }
            else
            {  if ( amt > 1 )
               {  next_st = 2;
                  if ( cur_st == 2 ) gap_q22 = gap_q22 + amt – (long)1;
               }
            }
            cur_st = next_st;
         }
         fclose(fpin);

         q21_ser = (float)gap_q21/((float)gap_q21+(float)gap_q22);
         q12_ser = cpt * q21_ser / (1.0 – cpt);

         if ( q12_ser > 0 && q12_ser < 1.0 )
         {  j = 1;
            l = 1;
            ssq_cpt = 0;
            while ( j < 6 )
            {  k = 1;
               while ( k < 10 )
               {  x = (float)k * pow(10.0,(float)j)/10.0;
                  erf_cpt = ((1.0–cpt)*pow((1.0–e1_cpt),x)) + (cpt*pow((1.0–e2_ser),x));
```

```
            ssq_cpt = ssq_cpt +pow((erf_cpt-erfr[l]),2.0);
            ++l;
            ++k;
          }
          ++j;
        }
    }
    else ssq_cpt = 10000.0;
}
else ssq_cpt = 10000.0;

while ( distance >= 0.00000001 )
{   lpt = cpt – distance;
    rpt = cpt + distance;
    e1_lpt = ber*e2_ser*(1.0–lpt)/(e2_ser–(ber*lpt));
    e1_rpt = ber*e2_ser*(1.0–rpt)/(e2_ser–(ber*rpt));
    if ( e1_lpt > 0 )
    {   x = ( log(0.05) / log(1.0–e2_ser) ) + 0.5;
        mx_gpst2 = (long)x;
        gap_q21 = gap_q22 = 0;

        fpin = fopen("dsto_gap.dat", "r");

        fscanf(fpin, "%u %u", &sig_nal, &amt);
        if ( sig_nal == 0 )
        {   if ( amt > mx_gpst2 ) cur_st = 1;
            else cur_st = 2;
        }
        while (.fscanf(fpin, "%u %u", &sig_nal, &amt) != EOF )
        {   if ( sig_nal == 0 )
            {   if ( amt > mx_gpst2 )
                {   next_st =1;
                    if ( cur_st == 2 ) ++gap_q21;
                }
                else
                {   next_st = 2;
                    if ( cur_st == 2 ) ++gap_q22;
                }
            }
            else
            {   if ( amt > 1 )
                {   next_st = 2;
                    if ( cur_st == 2 ) gap_q22 = gap_q22 + amt – (long)1;
                }
            }
            cur_st = next_st;
        }
        fclose(fpin);

        q21_ser = (float)gap_q21 / ( (float)gap_q21 + (float)gap_q22 );
        q12_ser = lpt * q21_ser / ( 1.0 – lpt );

        if ( q12_ser > 0 && q12_ser < 1.0 )
        {   j = 1;
            l = 1;
```

```
      ssq_lpt = 0;
      while ( j < 6 )
      {   k = 1;
          while ( k < 10 )
          {   x = (float)k * pow(10.0,(float)j)/10.0;
              crf_lpt = ((1.0-lpt)*pow((1.0-c1_lpt),x)) + (lpt*pow((1.0-c2_ser),x));
              ssq_lpt = ssq_lpt +pow((crf_lpt-crfr[l]),2.0);
              ++l;
              ++k;
          }
          ++j;
      }
    }
    else ssq_lpt = 10000.0;
}
else ssq_lpt = 10000.0;

if ( e1_rpt > 0 )
{   x = ( log(0.05) / log(1.0-e2_ser) ) + 0.5;
    mx_gpst2 = (long)x;
    gap_q21 = gap_q22 = 0;

    fpin = fopen("dsto_gap.dat", "r");

    fscanf(fpin, "%u %u", &sig_nal, &amt);
    if ( sig_nal == 0 )
    {   if ( amt > mx_gpst2 ) cur_st = 1;
        else cur_st = 2;
    }
    while ( fscanf(fpin, "%u %u", &sig_nal, &amt) != EOF )
    {   if ( sig_nal == 0 )
        {   if ( amt > mx_gpst2 )
            {   next_st =1;
                if ( cur_st == 2 ) ++gap_q21;
            }
            else
            {   next_st = 2;
                if ( cur_st == 2 ) ++gap_q22;
            }
        }
        else
        {   if ( amt > 1 )
            {   next_st = 2;
                if ( cur_st == 2 ) gap_q22 = gap_q22 + amt - (long)1;
            }
        }
        cur_st = next_st;
    }
    fclose(fpin);

    q21_ser = (float)gap_q21 / ( (float)gap_q21 + (float)gap_q22 );
    q12_ser = rpt * q21_ser / ( 1.0 - rpt );

    if ( q12_ser > 0 && q12_ser < 1.0 )
    {   j = 1;
```

```
            l = 1;
            ssq_rpt = 0;
            while ( j < 6 )
            {   k = 1;
              while ( k < 10 )
              {   x = (float)k * pow(10.0,(float)j)/10.0;
                erf_rpt = ((1.0–rpt)*pow((1.0–e1_rpt),x)) + (rpt*pow((1.0–e2_ser),x));
                ssq_rpt = ssq_rpt +pow((erf_rpt–erfr[l]),2.0);
                ++l;
                ++k;
              }
              ++j;
            }
          }
          else ssq_rpt = 10000.0;
        }
        else ssq_rpt = 10000.0;

        if ( ssq_lpt <= ssq_cpt && ssq_lpt < ssq_rpt )
        {   cpt = lpt;
          ssq_cpt = ssq_lpt;
        }
        else if ( ssq_rpt <= ssq_cpt && ssq_rpt < ssq_lpt )
        {   cpt = rpt;
          ssq_cpt = ssq_rpt;
        }
        else
        {   cpt = cpt;
          ssq_cpt = ssq_cpt;
        }
        distance = distance / 2.0;
      }

      if ( i == 1 )
      {   cent_ssq = ssq_cpt;
        cent_q2 = cpt;
        cent_e2 = e2_ser;
      }
      if ( i == 2 )
      {   left_ssq = ssq_cpt;
        left_q2 = cpt;
        left_e2 = e2_ser;
      }
      if ( i == 3 )
      {   right_ssq = ssq_cpt;
        right_q2 = cpt;
        right_e2 = e2_ser;
      }
    }

    if ( left_ssq <= cent_ssq && left_ssq < right_ssq )
    {   center = left;
      cent_e2 = left_e2;
      cent_q2 = left_q2;
      cent_ssq = left_ssq;
```

```
  }
  else if ( right_ssq <= cent_ssq && right_ssq < left_ssq )
  {  center = right;
     cent_e2 = right_e2;
     cent_q2 = right_q2;
     cent_ssq = right_ssq;
  }
  else
  {  center = center;
     cent_e2 = cent_e2;
     cent_q2 = cent_q2;
     cent_ssq = cent_ssq;
  }
  dist_out = dist_out/2.0;
}

e2 = cent_e2;
q2 = cent_q2;
q1 = 1.0 – q2;
e1 = ber * e2 * q1 / ( e2 – (ber * q2) );
tot_ssq = cent_ssq;

x = ( log(0.05) / log(1.0–e2) ) + 0.5;
mx_gpst2 = (long)x;
gap_q21 = gap_q22 = 0;

fpin = fopen("dsto_gap.dat", "r");

fscanf(fpin, "%u %u", &sig_nal, &amt);
if ( sig_nal == 0 )
{  if ( amt > mx_gpst2 ) cur_st = 1;
   else cur_st = 2;
}
while ( fscanf(fpin, "%u %u", &sig_nal, &amt) != EOF )
{  if ( sig_nal == 0 )
   {  if ( amt > mx_gpst2 )
      {  next_st =1;
         if ( cur_st == 2 ) ++gap_q21;
      }
      else
      {  next_st = 2;
         if ( cur_st == 2 ) ++gap_q22;
      }
   }
   else
   {  if ( amt > 1 )
      {  next_st = 2;
         if ( cur_st == 2 ) gap_q22 = gap_q22 + amt – (long)1;
      }
   }
   cur_st = next_st;
}
fclose(fpin);

q21 = (float)gap_q21 / ( (float)gap_q21 + (float)gap_q22 );
```

```
q22 = 1.0 – q21;
q12 = q2 * q21 / q1;
q11 = 1.0 – q12;
ber_cal = 1.0 / ( ( q1/e1 ) + ( q2/e2 ) );

fprintf(fpout, "Binary search technique : \n\n");
fprintf(fpout, "The actual BER     = %9.8f \n\n", ber);

fprintf(fpout, "The parameters of 2–state Markov Chain Model are as following : \n\n");
fprintf(fpout, "The calculated BER =  %9.8f \n", ber_cal);
fprintf(fpout, "The BER in good state ( e1 ) = %9.8f \n", e1);
fprintf(fpout, "The BER in bad  state ( e2 ) = %9.8f \n", e2);
fprintf(fpout, "The transition probability from good state to itself ( q11 ) = %9.8f \n", q11);
fprintf(fpout, "The transition probability from good to bad state   ( q12 ) = %9.8f \n", q12);
fprintf(fpout, "The transition probability from bad state to itself  ( q22 ) = %9.8f \n", q22);
fprintf(fpout, "The transition probability from bad to good state   ( q21 ) = %9.8f \n", q21);
fprintf(fpout, "The probability of being in good state after error  ( Q1 ) = %9.8f \n", q1);
fprintf(fpout, "The probability of being in bad state after error   ( Q2 ) = %9.8f \n", q2);

p1 = ber_cal * q1 / e1;
p2 = 1.0 – p1;

fprintf(fpout, "The unconditional probability of being in good state ( R1 ) = %9.8f \n", p1);
fprintf(fpout, "The unconditional probability of being in bad state  ( R2 ) = %9.8f \n\n", p2);
fprintf(fpout, "SSQ =  %9.8f \n", tot_ssq);

fclose(fpout);
}
```

"erfr_cal.c"

is the program to calculate the error–free run distribution of the model using the model parameters obtained from the optimisation method.

Note: Before running the program, one output filename and four variables are essentially given. In the listing attached, an example of the output filename is **"dsto_meth2.erf_cal"** and the four variables are q_1, q_2, e_1 and e_2 (the model parameters).

มหาวิทยาลัยเชียงใหม่ Chiang Mai University

```
/*    Program to printout the calculated        *
 *       Error-free Run Distribution            */


#include <stdio.h>
#include <math.h>


FILE *fpout;


main()
{  int i, j, m;
   float erfr, q1, q2, e1, e2;
   long k;
   fpout = fopen("dsto_meth2.erf_cal", "a");
   i = j = m = 1;
   q1 = 0.12269402;
   q2 = 0.87730598;
   e1 = 0.00478297;
   e2 = 0.03946850;
   while ( i < 6 )
   {  while ( j < 10 )
      {  k = (long)j * (long)pow(10.0,(float)i) / (long)10;
         erfr = (q1*pow((1.0-e1),(float)k)) + (q2*pow((1.0-e2),(float)k));
         fprintf(fpout, "%15u %20.6f\n", k, erfr);
         ++j;
      }
      ++i;
      j = 1;
   }
}
```

มหาวิทยาลัยเชียงใหม่ Chiang Mai University

"pmn_cal.c"

is the program to calculate the block error statistics of the model using equation in Chapter 3 and the model parameters obtained from the optimisation method.

Note: Before running the program, one output filename and four variables are essentially given. In the listing attached, an example of the output filename is "pmn_cal.real" and the four variables are q_{12}, q_{21}, e_1 and e_2 (the model parameters).

มหาวิทยาลัยเชียงใหม่ Chiang Mai University

```
/*-------------------------------------------------------------*
 *                                                             *
 *    Program to calculate P(m,n)                              *
 *      for 2-state Markov chain model                         *
 *      ( McCullough's model                                   *
 *                                                             *
 *            Date  :                                          *
 *            By    : Adisak Jaisilp                           *
 *                                                             *
 *-------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>

FILE *fpout;

main()
{  int m, n, mm, nn;
   float e1, e2, q1, q2, r1, r2, av_ber;
   float q11, q12, q22, q21;
   float a1[8][32], a2[8][32], p[8][32];

   fpout = fopen("pmn_cal.real", "a");

   e1 = 0.004783;
   e2 = 0.0394685;
   q12 = 0.66278541;
   q21 = 0.09269264;

   q11 = 1.0 - q12;
   q22 = 1.0 - q21;
   q1 = q21 / ( q12 + q21 );
   q2 = 1.0 - q1;
   av_ber = 1.0 / ((q1/e1)+(q2/e2));
   r1 = av_ber * q1/e1;
   r2 = 1.0 - r1;

   p[1][1] = av_ber;
   a1[1][1] = (r1*e1*q11)+(r2*e2*q21);
   a2[1][1] = p[1][1] - a1[1][1];

   n = 1;
   while ( n <= 30 )
   {  a1[0][n] = r1*pow((1.0-e1),(float)n);
      a2[0][n] = r2*pow((1.0-e2),(float)n);
      p[0][n]  = a1[0][n] + a2[0][n];
      ++n;
   }

   n = 2;
   while ( n <= 6 )
   {  nn = n - 1;
      a1[n][n] = (a1[nn][nn]*e1*q11)+(a2[nn][nn]*e2*q21);
      a2[n][n] = (a2[nn][nn]*e2*q22)+(a1[nn][nn]*e1*q12);
      p[n][n]  = a1[n][n] + a2[n][n];
      ++n;
   }
```

```
    m = 1;
    while ( m <= 6 )
    {   n = m + 1;
        while ( n <= 30 )
        {   nn = n - 1;
            mm = m - 1;
            a1[m][n] = (a1[m][nn]*(1.0-c1))+(a1[mm][nn]*c1*q11)+(a2[mm][nn]*c2*q21);
            a2[m][n] = (a2[m][nn]*(1.0-c2))+(a2[mm][nn]*c2*q22)+(a1[mm][nn]*c1*q12);
            p[m][n]  = a1[m][n] + a2[m][n];
            ++n;
        }
        ++m;
    }

    fprintf(fpout, "#n     m      P(m,n) \n");
    n = 1;
    while ( n <= 30 )
    {   m = 0;
        while ( m <= n && m <= 5 )
        {   fprintf(fpout, "%2u    %2u    %9.8f \n",n, m, p[m][n]);
            ++m;
        }
        ++n;
    }
    fclose(fpout);
}
```

มหาวิทยาลัยเชียงใหม่ Chiang Mai University

"datsmgen.c"

is the program to generate simulated data using the 2–state Markov chain model with parameters obtained from the optimisation method. The output of this program is a {0,1} sequence whose error characteristics can then be examined in the same way as the original or interleaved data.

Note: Before running the program, one output filename and four variables are essentially given. In the listing attached, an example of the output filename is **"dsto_sim1.dat"** and the four variables are q_{12}, q_{21}, e_1 and e_2 (the model parameters).

มหาวิทยาลัยเชียงใหม่ Chiang Mai University

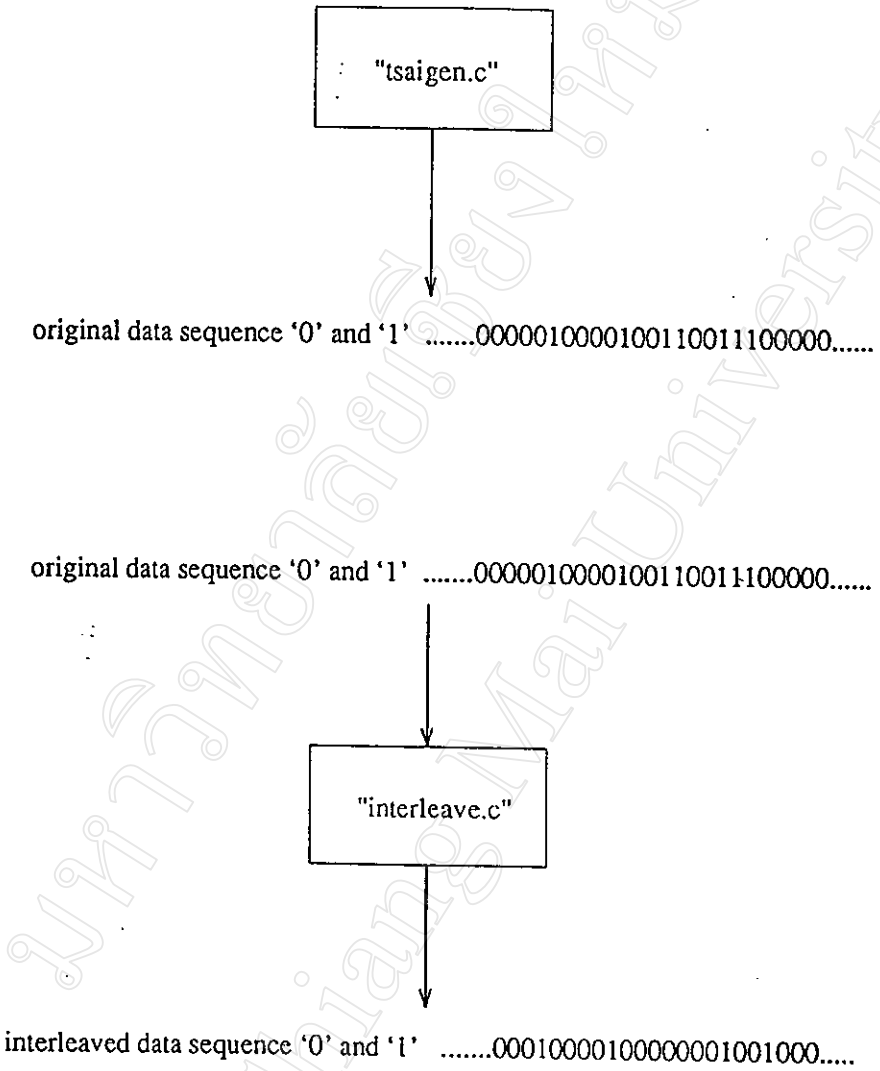
```
/*-------------------------------------------------------------*
*                                                             *
*   Program to generate error sequence by using               *
*     2-state Markov Chain Model with the parameters          *
*       from the optimization technique                       *
*                                                             *
*   Date :                                                    *
*   By   : Adisak  Jaisilp                                    *
*                                                             *
*-------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MAXDAT 4000000

FILE *fpout;

float rannum();

main()
{  int state, dat_bit;
   long iter;
   float q12, q21, e1, e2;
   fpout = fopen("dsto_sim1.dat", "a");
   state = 1;
   iter = 0;
   e1  = 0.060901914;
   e2  = 0.030817812;
   q12 = 0.789473653;
   q21 = 0.022838056;
   while ( iter < MAXDAT )
   {  if ( state == 1 )
      {  if ( rannum() > e1 ) dat_bit = '0';
         else
         {  dat_bit = '1';
            if ( rannum() < q12 ) state = 2;
         }
      }
      else
      {  if ( rannum() > e2 ) dat_bit = '0';
         else
         {  dat_bit = '1';
            if ( rannum() < q21 ) state = 1;
         }
      }
      fputc(dat_bit, fpout);
      ++iter;
   }
   fclose(fpout);
}

float rannum()
{  float a1, a2, a3, a4, r;
   int a5;
```

```
    a1 = rand()/pow(2.0, 31.0);
    a2 = rand()/pow(2.0, 31.0);
    a3 = rand()/pow(2.0, 31.0);
    a4 = a1 + a2 + a3;
    a5 = a4;
    r = a4 – a5;
    return r;
}
```

มหาวิทยาลัยเชียงใหม่
Chiang Mai University

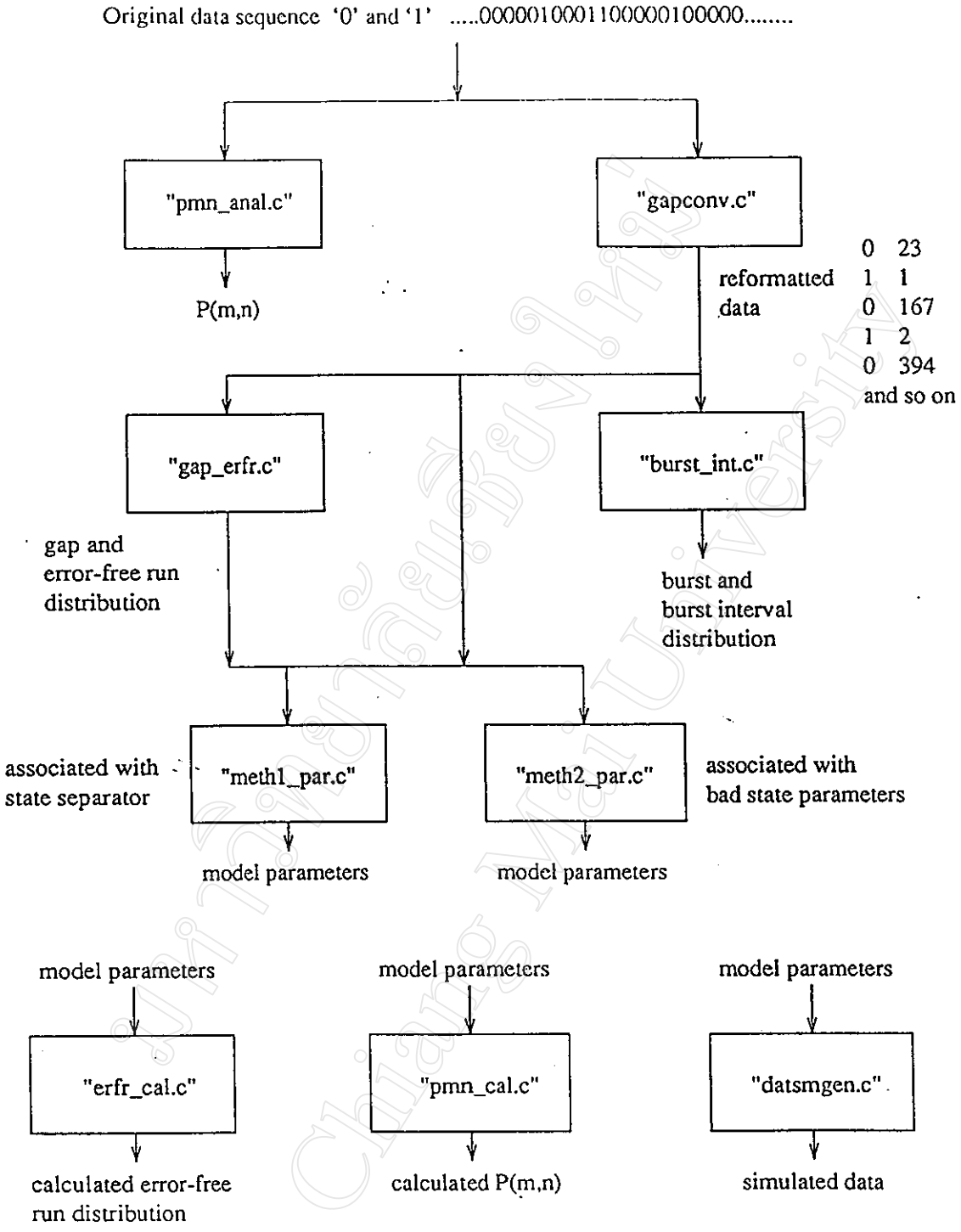
2. Program Structure

The program processing sequence for channel modelling is shown below.



continue.....

มหาวิทยาลัยเชียงใหม่ Chiang Mai University


Original data sequence '0' and '1' .....0000010001100000100000.....
"pmn_anal.c"
"gapconv.c"
P(m,n)
reformatted data
0 23
1 1
0 167
1 2
0 394
and so on
"gap_erfr.c"
"burst_int.c"
gap and error-free run distribution
burst and burst interval distribution
associated with state separator
"meth1_par.c"
"meth2_par.c"
associated with bad state parameters
model parameters
model parameters
model parameters
model parameters
model parameters
"erfr_cal.c"
"pmn_cal.c"
"datsmgen.c"
calculated error-free run distribution
calculated P(m,n)
simulated data

ภาคผนวก ค

สิ่งตีพิมพ์เผยแพร่ที่เกิดจากงานวิจัยนี้

[1] "การจำลองช่องสื่อสารชนิดความผิดพลาดแบบเบิรสต์
โดยใช้แบบจำลองลูกโซ่มาร์คอฟสองสถานะ"
"Burst-Error-Channel Modelling using 2-state
Markov Chain Model"

งานประชุมวิชาการวิศวกรรมไฟฟ้า
ครั้งที่ 20 พ.ศ. 2540 กรุงเทพมหานคร

[2] "ค่าความลึกที่เหมาะสมของการสลับตำแหน่งข้อมูล
ในการทำให้ความผิดพลาดแบบเบิรสต์เป็นแบบสุ่ม"
"Optimum Depth of Data Interleaving to
Randomize Burst-Errors"

วารสารวิศวกรรมศาสตร์ มหาวิทยาลัยเชียงใหม่
Vol. 8. January 2000

มหาวิทยาลัยเชียงใหม่ Chiang Mai University

ประวัติผู้เขียน

ชื่อ	นายอดิศักดิ์ ใจศิลป์
วัน เดือน ปี เกิด	8 พฤษภาคม 2503
ประวัติการศึกษา	สำเร็จการศึกษามัธยมศึกษาตอนปลาย โรงเรียนปรินส์รอยแยลส์ วิทยาลัย เชียงใหม่ ปีการศึกษา 2520 สำเร็จการศึกษาปริญญาวิศวกรรมศาสตร์บัณฑิต สาขาวิศวกรรมไฟฟ้า มหาวิทยาลัยเชียงใหม่ ปีการศึกษา 2524 สำเร็จการศึกษาระดับ Graduate Diploma in Electronic Systems จาก University of South Australia, Australia 1992.
ทุนการศึกษา	ได้รับทุนอุดหนุนและส่งเสริมวิทยานิพนธ์ระดับปริญญาโท บัณฑิตวิทยาลัย มหาวิทยาลัยเชียงใหม่
ประสบการณ์	วิศวกร สำนักบริการคอมพิวเตอร์ มหาวิทยาลัยเชียงใหม่ ปี พ.ศ. 2525 – 2528 อาจารย์ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเชียงใหม่ ปี พ.ศ. 2529 - ปัจจุบัน

มหาวิทยาลัยเชียงใหม่ Chiang Mai University