ms2dna, v. 1.16: Convert Simulated Haplotype Data to DNA Sequences

Bernhard Haubold & Peter Pfaffelhuber

September 18, 2013

Abstract

ms2dna is a program for converting haplotypes generated by the popular coalescent simulation programs ms or macs to DNA sequence data.

Introduction

When testing DNA sequence analysis tools, it is often desirable to simulate sequence samples that have evolved under well defined scenarios. A simple method to achieve this is to convert the haplotypes generated by available coalescent simulators to DNA sequences. Perhaps the most widely used coalescent simulator is ms and its user interface an output format have become de facto standards in the field [2]. Among the alternatives macs is particularly interesting as it can simulate long sequences with realistic recombination rates [1]. Output generated by macs can be converted to ms-format using msformatter. Hence I restrict my description to the output of ms, a sample of which is stored in the file simpleTest.dat and looks like this:

```
ms 3 2 -t 10 -r 3 1000
52440 47412 34298

//
segsites: 4
positions: 0.1544 0.1855 0.2341 0.7006
1110
0110
0001

//
segsites: 7
positions: 0.2051 0.3380 0.3761 0.3982 0.4218 0.7364 0.8214
0000100
1111000
0000011
```

The first line repeats the original command with which ms was run. In this case it means that two samples of size three are generated with neutral mutation parameter $\theta = 4N\mu = 10$ and recombination parameter $R = 4N\rho = 3$ acting on a sequence of 1000 recombining loci; we can think of these as 1000 bp. The result consists of two sets of three sequences. The first set has 4 segregating sites (mutations), the second 7. These mutations are modeled as falling onto the interval between 0 and 1, in the first data set they occur at positions $0.1544, 0.1855, 0.2341, 0.7006$. Each row of zeros and ones represents a haplotype, where we can interpret a zero as the wild type and a 1 as the mutant genotype.

The purpose of ms2dna is to convert the haplotypes generated by ms into the corresponding DNA sequences of a certain length. There already exists a freely available program that is often used for this purpose: seq-gen [3] takes as input a genealogy, which might have been produced by ms, and generates a compatible sequence sample. In contrast, ms2dna directly converts the haplotypes generated by ms to DNA sequence data. This may be

advantageous in situations where the sample is generated by an ancestral recombination graph rather than by a tree.

Compilation

The program `ms2dna` is distributed as a compressed archive containing C-source code, test data and documentation. The software is developed on a computer running the Linux operating system and the following instructions for compiling and using the program refer to a UNIX-type system. Compilation on any other platform should be simple as long as a C-compiler is available. In order to generate the program from the source distribution, unpack the archive:

```
tar -xvzf ms2dna_XXX.tgz
```

where `XXX` indicates the current program. Change into the directory thus generated

```
cd Ms2dna_XXX
```

and compile the program

```
make
```

This generates the executable `ms2dna`, which can now be run

```
./ms2dna -h
```

to return the options list. In order to convert the supplied simple test data set execute the command

```
./ms2dna < simpleTest.dat
```

Alternatively, the mutations specified by the test data can be applied to an external ancestral sequence:

```
./ms2dna -t template.fasta < simpleTest.dat
```

Program Details

Input

`ms2dna` takes two kinds of input:

1. Haplotype information generated by `ms`, and optionally a
2. template sequence that gets mutated.

By default `ms2dna` reads the haplotype information from the standard input (STDIN); it can also read it from one or more files listed after the options. The optional template sequence is read from the file specified by the `-t` option. This is expected to contain a single DNA sequence in FASTA format of length equal to the number of residues simulated by `ms` and containing exclusively the characters `[acgtACGT]`.

The program starts by reading parameters from the command line echoed by `ms`. In particular, it reads the length of the DNA sequence from the second argument of the `-r` switch. For recombination-free data run `ms` with 0 as the first argument of `-r`, e.g.

```
ms 3 1 -t 10 -r 0 1000 | ms2dna
```

generates three recombination-free sequences of one kb length. Similar output can be generated with `macs` using the command

```
macs 3 1000 -t 0.01 2> /dev/null | msformatter | ms2dna -a
```

Output

The output is printed to the standard output stream (STDOUT). The output is by default in FASTA format. Alternatively, the `-f` switch can be used to request my very own "population genetics format" (pgf), which is like FASTA except that the first line consists of three numbers specifying

1. number of taxa
2. number of nucleotides
3. starting position

Each sample generated by `ms` is interpreted as representing data from an independent locus. This is also reflected in the sequence headers where the y-th sequence of the x-th locus would be identified as

```
>LX_SY
```

The example data contained in the file `simpleTest.dat` consists of two loci from which three sequences were sampled each.

The file `migrationTest.dat` contains simulated data drawn from a structured population consisting of two subpopulations. For the interpretation of such data it is often convenient to include population information in the sequence identifier. The z-th sequence from the y-th population at the x-th locus is therefore labeled by `ms2dna`

```
>LX_PY_SZ
```

Mutation Model

`ms` implements the infinite sites model, i. e. no position in a sequence can mutate twice. This is guaranteed by representing the positions of mutations as random real numbers. With discrete sequences a given position in a DNA sequence may be hit twice. In this case, `ms2dna` draws a new random position. Alternatively, if the option `-m` is used `ms` looks the closest hitherto unmutated position in the neighborhood. In case the number of segregating sites exceeds the number of mutable positions, `ms2dna` prints a warning, e.g.:

```
#WARNING: number of segregating sites (1001) > number of mutable sites (1000)
```

However, as long as the sequence length is much larger than the expected number of segregating sites, this should happen very rarely.

For each sample parsed by `ms2dna` a random ancestral sequence is either generated from scratch or read from the template file (if specified). The ancestral sequence is then mutated at the positions output by `ms`. The G/C content can be set using the `-g` switch. Under the default G/C content (0.5), all nucleotides mutate with equal probability into the other three nucleotides. If the G/C content $\neq 0.5$, this affects both the ancestral as well as in the mutated positions (as expected), unless a template sequence is supplied. In that case the G/C content affects only the spectrum of derived alleles.

Random Number Generator

The sequences generated by `ms2dna` are, of course, random. As a result, if `ms2dna` is run repeatedly on the same input data, different sequences are generated every time. In order to guarantee this behavior, while still allowing the user to switch it off conveniently, `ms2dna` looks for a seed from three different sources in the following order

1. the argument to the `-s` switch; if this is not supplied (default),
2. the number contained in the file `randomSeed.dat`; if this does not exist,
3. the system clock.

At the end of a run a seed for the next run is stored in `randomSeed.dat`.

Listings

This section contains listings of the code for the central parts of `ms2dna`.

0.1 The Driver Program: `ms2dna.c`

```
/***** ms2dna.c *************************************
 * Description: Convert haplotypes from Hudons's
 *   ms program to DNA sequence data.
 * Reference: Hudson, R. R. (2002). Generating
 *   samples under a Wright-Fisher neutral model
 *   of genetic variation. Bioinformatics 18: 337-338.
 * Author: Bernhard Haubold, haubold@evolbio.mpg.de
 * Date: Tue Aug 14 15:15:47 2007.
 * License: GNU General Public
 ****************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include "eprintf.h"
#include "interface.h"
#include "stringUtil.h"
#include "sample.h"
#include "ran.h"

int runAnalysis(FILE *fpin, Args *args);

int main(int argc, char *argv[]){
  char *version;
  Args *args;
  FILE *fpin;
  int i;

  version = "1.16";
  setprogname2("ms2dna");
  args = getArgs(argc, argv);

  if(args->h == 1){
    printUsage(version);
    return 0;
  }else if(args->e == 1){
    printUsage(version);
    return -1;
  }else if(args->p){
    printSplash(version);
    return 0;
  }

  if(args->numInputFiles == 0){
    fpin = stdin;
    runAnalysis(fpin, args);
  }else{
```

```
      for(i=0;i<args->numInputFiles;i++){
        fpin = efopen(args->inputFiles[i],"r");
        runAnalysis(fpin, args);
        fclose(fpin);
      }
    }
    free(args);
    free(progname());
    return 0;
}

int runAnalysis(FILE *fpin, Args *args){
    Sample *sample;
    FILE *fpout = stdout;

    sample = initializeSample(fpin, args);
    if(args->f == 1 && sample->npop != 2){
      printf("ERROR[ms2dna]: need two populations for ima output format\n");
      return -1;
    }
    if(args->f == -1)
      fprintf(fpout,"%d %d 1\n",sample->nsam*sample->howmany,sample->nsite);
    if(args->f == 1){
      fprintf(fpout,"Sample generated by ms2dna\n");
      fprintf(fpout,"#Comment\n");
      fprintf(fpout,"P1 P2\n");
      fprintf(fpout,"%d\n",sample->howmany);
    }
    while((sample = getSample(args->a)) != NULL){
      outputSample(fpout);
    }
    freeSample();
    return 0;
}
```

0.2 Sample Processing: `sample.c`

```
/***** sample.c ************************************
 * Description: Functions for manipulating a sample
 *   generated by Hudson's ms program.
 * Reference: Hudson, R. R. (2002). Generating samples
 *   under a Wright-Fisher neutral model of genetic
 *   variation. Bioinformatics 18: 337-338.
 * Author: Bernhard Haubold, haubold@evolbio.mpg.de
 * Date: Tue Aug 14 20:35:23 2007.
 * License: GNU General Public
 ***************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <math.h>
#include <unistd.h>
#include <fcntl.h>
#include "eprintf.h"
```

```
#include "interface.h"
#include "sample.h"
#include "ran.h"
#include "sequence_data.h"
#include "stringUtil.h"

#define MAXLEN 1000

Sample *sample;
Args *args;
FILE *fp;
int sampleCounter;

void expandSample();
void getTemplate(char *file);

int comp(const void *v1, const void *v2){
     return (*(int *)v1 - *(int *)v2);
}


Sample *initializeSample(FILE *filePointer, Args *arguments){
  char *token;
  int i, idum;
  FILE *fpra;

  args = arguments;
  fp = filePointer;
  sampleCounter = 0;

  sample = (Sample *)emalloc(sizeof(Sample));
  sample->line = (char *)emalloc((MAXLEN+1) * sizeof(char));
  sample->line[0] = '\0';
  /* get first line of input; this contains the command with which ms was
     started */
  sample->line = fgets(sample->line, MAXLEN, fp);
  token = strtok(sample->line, " ");
  token = strtok(NULL, " ");
  sample->nsam = atoi(token);
  if(args->a){
    token = strtok(NULL, " ");
    sample->nsite = atoi(token);
    sample->howmany = 1;
  }else{
    token = strtok(NULL, " ");
    sample->howmany = atoi(token);
  }
  sample->npop = 1;
  sample->sampleSizes = NULL;
  while((token = strtok(NULL, " ")) != NULL){
    if(strcmp(token, "-r") == 0){
      token = strtok(NULL, " ");
      if(!args->a){
        token = strtok(NULL, " ");
```

```
        sample->nsite = atoi(token);
      }
    }else if(strcmp(token, "-I") == 0){
      token = strtok(NULL, " ");
      sample->npop = atoi(token);
      sample->sampleSizes = (int *)emalloc(sample->npop*sizeof(int));
      for(i=0;i<sample->npop;i++){
        token = strtok(NULL, " ");
        sample->sampleSizes[i] = atoi(token);
      }
    }
  }
  if(sample->nsite == 0 && args->a){
    fprintf(stderr,"ERROR [sample.c]: please use ms with the -r switch.\n")
       ;
    exit(-1);
  }
  sample->seq = (char **)emalloc(2 * sizeof(char *));
  if(args->t)
    getTemplate(args->t);
  else{
    sample->seq[0] = (char *)emalloc((sample->nsite + 1) * sizeof(char));
    sample->seq[1] = (char *)emalloc((sample->nsite + 1) * sizeof(char));
  }
  sample->maxlen = 1;
  sample->segsites = 0;
  sample->haplotypes = (char **)emalloc((sample->nsam + 1)*sizeof(char *));
  for(i=0;i<sample->nsam;i++){
    sample->haplotypes[i] = (char *)emalloc((sample->maxlen + 1)*sizeof(
       char));
  }
  sample->map = (int *)emalloc(sample->maxlen * sizeof(int));
  sample->positions = (float *)emalloc(sample->maxlen*sizeof(float));

  /* seed and initialize random number generator */
  if(args->s != 0){
    idum = args->s;
  }else if((fpra = fopen("randomSeed.dat","r")) != NULL){
    if(!fscanf(fpra,"%d",&idum))
      printf("WARNING[sample.initializeSample]: Something is wrong writing
         the the seed of the random number generator to file.\n");
    fclose(fpra);
  }else
    idum = -time(NULL);
  init_genrand(idum);
  return sample;
}

void getTemplate(char *file){
  int fd;
  Sequence *sequence;
  int *dic, i;

  if ((fd = open (file, O_RDONLY, 0)) == 0)
```

```
123       eprintf("ERROR [ms2dna]: cannot open template file %s for reading\n",
              file);
      sequence = read_fasta(fd);
      close(fd);

      if(sequence->num_seq > 1){
128     printf("ERROR[ms2dna]: the template file contains %d sequences\n",
              sequence->num_seq);
        printf("\tbut the program can only deal with template files containing
              a single sequence.\n");
        exit(0);
      }
      sequence->len--;
133   if(sequence->len != sample->nsite){
        printf("ERROR[ms2dna]: the template file contains %ld nucleotides,\n",
              sequence->len);
        printf("\tbut the ms simulation deals with %d sites.\n",sample->nsite);
        printf("\tthese two numbers need to be identical.\n");
        exit(0);
138   }
      dic = NULL;
      dic = get_restricted_dna_dictionary(dic);
      for(i=0;i<sequence->len;i++)
        if(!dic[(int)sequence->seq[i]]){
143       printf("ERROR[ms2dna]: the template sequence contains residues other
              than [acgtACGT] at position %d: %c.\n",i,sequence->seq[i]);
          exit(0);
        }
      strtoupper(sequence->seq);
      sample->seq[0] = sequence->seq;
148   sample->seq[1] = (char *)emalloc((sample->nsite + 1) * sizeof(char));
    }

    /* getSample: read a haplotype sample from a file pointer assumed to be
       open */
    Sample *getSample(int isMacs){
153   int i;
      char *dum;
      double r1, r2;

      dum = emalloc(250*sizeof(char));
158   while(fgets(sample->line,MAXLEN,fp) != NULL){
        if(sample->line[0] == 's'){
          sscanf(sample->line, "segsites: %d", &sample->segsites);
          if(sample->segsites >= sample->maxlen){
            sample->maxlen = sample->segsites+1;
163         expandSample(sample);
          }
          if(!fscanf(fp,"%s",dum)){
            printf("ERROR[sample.getSample()]: Cannot read seed for random
                number generator from file.\n");
            exit(-1);
168       }
          if(sample->segsites > 0){
```

```
        for(i=0;i<sample->segsites;i++)
          if(!fscanf(fp, "%f", sample->positions + i))
            printf("WARNING[smple.getSample()]-1: Something is wrong with
                reading the sample.\n");
        if(!isMacs){
          /* generate extra significant digits for ms output, as ms only
              generates 4 */
          for(i=0;i<sample->segsites;i++){
            r1 = genrand_real1();
            r2 = genrand_real1();
            r2 /= 10000;
            if(r1>0.5 && sample->positions[i] - r2 >= 0)
              sample->positions[i] -= r2;
            else if(sample->positions[i] + r2 <= 1.0)
              sample->positions[i] += r2;
          }
        }
        for(i=0;i<sample->nsam;i++){
          if(!fscanf(fp," %s",sample->haplotypes[i]))
            printf("WARNING[smple.getSample()]-2: Something is wrong with
                reading the sample.\n");
        }
      }
      free(dum);
      return sample;
    }
  }
  free(dum);
  return NULL;
}

void outputSample(FILE *fpo){
  int i, j, stepsRight, stepsLeft;
  double r1, r2;
  int p;
  char nuc;

  if(!args->t){
    /* generate ancestral sequence */
    for(i=0;i<sample->nsite;i++){
      r1 = genrand_real1();
      r2 = genrand_real1();
      if(r1 <= args->g){
        if(r2 <= 0.5)
          sample->seq[0][i] = 'G';
        else
          sample->seq[0][i] = 'C';
      }else{
        if(r2 <= 0.5)
          sample->seq[0][i] = 'A';
        else
          sample->seq[0][i] = 'T';
      }
    }
```

```
  }
  for(i=0;i<sample->nsite;i++)
    sample->seq[1][i] = 'x';
  if(sample->segsites > sample->nsite){
    printf("#WARNING: number of segregating sites (%d) > number of mutable
        sites (%d)\n",sample->segsites,sample->nsite);
    printf("#Are you dealing with macs input? If so, please use the -a
        option.\n");
  }
  /* map segregating sites onto sequence positions */
  for(i=0;i<sample->segsites;i++){
    p = sample->positions[i] * sample->nsite;
    while(sample->seq[1][p] != 'x'){ /* check for double hits */
      if(args->m == 0){
        p = genrand_real1() * sample->nsite;
      }else if(args->m == 1){
        stepsLeft = 0;
        stepsRight = 0;
        /* walk to the right */
        for(j=p+1;j<sample->nsite;j++){
          stepsRight++;
          if(sample->seq[1][j] == 'x')
            break;
        }
        /* walk to the left */
        j = (p-1 > 0) ? p-1 : 0;
        if(j){
          for(;j>=0;j--){
            stepsLeft++;
            if(sample->seq[1][j] == 'x')
              break;
          }
        }
        if((stepsLeft > 0) && (stepsLeft < stepsRight))
          p -= stepsLeft;
        else
          p += stepsRight;
      }
    }
    sample->map[i] = p;
    sample->seq[1][p] = 'y';
  }
  /* map the last segregating site to nonsense */
  sample->map[sample->segsites] = -1;
  /* generate nucleotides for mutant positions */
  for(i=0;i<sample->segsites;i++){
    r1 = genrand_real1();
    r2 = genrand_real1();
    p = sample->map[i];
    if(r1 <= args->g){                    /* is the derived state G/C? */
      if(sample->seq[0][p] == 'G')          /* is the ancestral state G? */
        nuc = 'C';                            /* mutate to C */
      else if(sample->seq[0][p] == 'C')     /* is the ancestral state C? */
        nuc = 'G';                            /* mutate to G */
```

```
273       else                                    /* is the ancestral state A/T?
              */
            if(r2 <= 0.5)                           /* draw equiprobable G or C
                */
              nuc = 'G';
            else
              nuc = 'C';
278       }else{                                /* is the derived state A/T? */
            if(sample->seq[0][p] == 'A')          /* is the ancestral state A? */
              nuc = 'T';                            /* mutate to T */
            else if(sample->seq[0][p] == 'T')     /* is the ancestral state T? */
              nuc = 'A';                            /* mutate to A */
283         else                                    /* is the ancestral state G/C?
              */
            if(r2 <= 0.5)                           /* draw equiprobable A or T
                */
              nuc = 'A';
            else
              nuc = 'T';
288       }
          sample->seq[1][p] = nuc;
        }
        sampleCounter++;
        if(args->f == 0)
293       printFasta(fpo);
        else if(args->f == 1)
          printIma(fpo);
        else if(args->f == 2)
          printGenetree(fpo);
298   }

      void printGenetree(FILE *fpo){

      }
303
      void printIma(FILE *fpo){
        int cc;  /* character chounter */
        int i, j, k;
        int ns;  /* number of segregating sites */
308     int pc;  /* population counter */
        int sc;  /* sequence counter (within population) */
        int maxLen = 10;

        pc = 0;
313     sc = 0;
        fprintf(fpo,"L_%d %d %d %d I 1\n",sampleCounter,
                sample->sampleSizes[0],sample->sampleSizes[1],sample->nsite);
        for(i=0;i<sample->nsam;i++){
          if(sc == sample->sampleSizes[pc]){
318         sc = 0;
            pc++;
          }
          sc++;
          sprintf(sample->line,"P%d_S%d",pc+1,sc);
```

```
      cc = strlen(sample->line) < maxLen ? strlen(sample->line) : maxLen;
      /* print 10 characters of haplotype label */
      for(j=0;j<cc;j++)
        fprintf(fpo,"%c",sample->line[j]);
      for(k=j;k<10;k++)
        fprintf(fpo,"%c",' ');
      /* print haplotype */
      ns = 0;
      for(j=0;j<sample->nsite;j++){
        if(j == sample->map[ns]){   /* print derived nucleotide */
          ns++;
          fprintf(fpo,"%c",sample->seq[1][j]);
        }else                       /* print ancestral nucleotide */
          fprintf(fpo,"%c",sample->seq[0][j]);
      }
      fprintf(fpo,"%s","\n");
    }
  }

  void printFasta(FILE *fpo){
    int ns;  /* segregating sites counter */
    int nc;  /* nucleotide counter */
    int sc;  /* sequence counter (within population) */
    int pc;  /* population counter */
    int i, j;
    int lineLen;

    lineLen = 70;
    nc = 0;
    sc = 0;
    pc = 0;
    for(i=0;i<sample->nsam;i++){
      fprintf(fpo,">");
      if(sample->howmany > 1)
        fprintf(fpo,"L%d",sampleCounter);
      if(sample->npop > 1){
        if(sample->nsam > 1)
          fprintf(fpo,"_P%d",pc+1);
        else
          fprintf(fpo,"P%d",pc+1);
      }
      if(sample->npop > 1 || sample->howmany > 1)
        fprintf(fpo,"_S%d\n",++sc);
      else
        fprintf(fpo,"S%d\n",++sc);
      if(sample->npop > 1){
        if(sc == sample->sampleSizes[pc]){
          sc = 0;
          pc++;
        }
      }
      /* print haplotype */
      nc = 0;
      ns = 0;
```

```
      qsort(sample->map,sample->segsites,sizeof(int),comp);
      for(j=0;j<sample->nsite;j++){
        if(ns >= sample->maxlen)
          printf("ns: %d; sample->maxlen: %d\n",ns,sample->maxlen);
        if(j == sample->map[ns]){ /* mutant position */
          if(sample->haplotypes[i][ns] == '1'){ /* print derived nucleotide
              */
            if(!args->c){
              fprintf(fpo,"%c",sample->seq[1][j]);
            }else{
              if((j+1)%3 == 0)
                fprintf(fpo,"%c",sample->seq[1][j]);
              else
                fprintf(fpo,"%c",sample->seq[0][j]);
            }
          }else                                 /* print ancestral nucleotide
              */
            fprintf(fpo,"%c",sample->seq[0][j]);
          ns++;
        }else
          fprintf(fpo,"%c",sample->seq[0][j]);
        nc++;
        if(nc == lineLen && j < sample->nsite - 1){
          fprintf(fpo,"\n");
          nc = 0;
        }
      }
/*     if(nc != lineLen) */
      fprintf(fpo,"\n");
    }
  }

  /* increase space for individual haplotypes
   */
  void expandSample(){
    int i;

    for(i=0;i<sample->nsam;i++)
      sample->haplotypes[i] = (char *)erealloc(sample->haplotypes[i],sample->
          maxlen*sizeof(char));
    sample->map = (int *)erealloc(sample->map,sample->maxlen*sizeof(int));
    sample->positions = (float *)erealloc(sample->positions,sample->maxlen*
        sizeof(float));

  }

  /* freeSample: clean up after run of program:
   * 1) free memory
   * 2) update seed for random number generator
   */
  void freeSample(){
    int i;
    FILE *fpra;
```

```
    for(i=0;i<sample->nsam;i++){
      free(sample->haplotypes[i]);
    }
    free(sample->haplotypes);
    free(sample->positions);
    free(sample->map);
    free(sample->seq[0]);
    free(sample->seq[1]);
    free(sample->seq);
    if(sample->sampleSizes)
      free(sample->sampleSizes);
    free(sample->line);
    free(sample);
    /* save seed of random number generator */
    if(args->s == 0){
      fpra = fopen("randomSeed.dat","w");
      fprintf(fpra,"%d\n",(int)genrand_int32());
      fclose(fpra);
    }
  }
```

Revision History

1. August 17, 2007: Version 0.1 produced
2. September 6 & 12, 2007: Version 0.2:
   - Included treatment of structured populations
3. October 5, 2007: Version 0.3
   - Fixed bug in treatment of multiple hits
4. October 27, 2007: Version 0.4
   - Allowed $S = 0$
5. March 29, 2008: Version 1.0 released
6. September 2008: Version 1.1
   - Included output in ima format
7. November 26, 2008: Version 1.2
   - Fixed treatment of multiple hits for FASTA format (ima format still needs to be checked): if some position, p, is hit repeatedly, the extra hit is moved to the next unmutated position starting from $p + 1$ and moving to the right while treating the sequence as circular.
8. April 14, 2009: Version 1.4
   - Fixed treatment of data sets where the number of nucleotides modulo the line length is zero (in this case the program inserted a superfluous carriage return).
9. April 21, 2009: Version 1.5
   - Changed mutation model (again): if a mutated position is hit again, the program looks for the closest unmutated position.
10. April 23, 2009: Version 1.6

- Fixed search for unmutated positions.

11. May 18, 2009: Version 1.7
    - Introduced `-m` switch to control treatment of double mutations.
12. June 5, 2009: Version 1.8
    - Introduced `-c` switch to mutate only third codon positions.
13. November 17, 2010: Version 1.9
    - Implemented `-t` option to allow input of a template sequence.
14. June 10, 2011: Version 1.10
    - Expanded position information to double precision.
15. September 1, 2011: Version 1.11
    - Implemented `-a` option to deal with input generated via `macs` — `msformatter` [1].
16. December 13, 2011: Version 1.12
    - Initialized last position of `sample->map` in line 249 of `sample.c`.
    - Removed minor memory leaks.
17. August 3, 2012: Version 1.13
    - Abolished addition of random significant digit for `macs` output. `ms` only prints 4 significant digits, which is problematic for long sequences, hence the addition of extra digits. However, `macs` returns six significant digits and addition of extra digits can lead to positions outside the interval $(0, 1)$.
    - Fixed handling of multiple samples generated by `macs`.
18. November 16, 2012: Version 1.14
    - Fixed addition of random significant digits for `ms` output such that no positions outside the interval $(0, 1)$ are generated.
19. September 17, 2013: Version 1.15
    - Fixed handling of `migrationTest.dat` by moving to initialization of `sample->sampleSizes` outside the `while` loop in `sample.initializeSample`.
20. September 18, 2013: Version 1.16
    - Fixed tokenization of the input line printed by `ms` in `sample.initializeSample`.

References

[1] G. K. Chen, P. Marjoram, and J. D. Wall. Fast and flexible simulation of DNA sequence data. *Genome Research*, 19:136–142, 2009.

[2] R. R. Hudson. Generating samples under a Wright-Fisher neutral model of genetic variation. *Bioinformatics*, 18:337–338, 2002.

[3] A. Rambaut and N. C. Grassly. Seq-gen: An application for the monte carlo simulation of dna sequence evolution along phylogenetic trees. *Bioinformatics*, 13:235–238, 1997.