`drawWrightFisher`, v. 1.3: Draw Wright-Fisher Population

Bernhard Haubold
Max-Planck-Institute for Evolutionary Biology, Plön, Germany

November 6, 2018

1 Introduction

The Wright-Fisher model is widely used in population genetics. The program `drawWrightFisher` is a teaching tool for visualizing this model.

2 Getting Started

`drawWrightFisher` was written in C on a computer running Linux and should work on any standard UNIX system. However, please contact me at `haubold@evolbio.mpg.de` if you have any problems with the program.

- Unpack the program

  ```
  tar -xvzf drawWrightFisher_XXX.tgz
  ```

  where `XXX` indicates the version.

- Change into the newly created directory

  ```
  cd DrawWrightFisher_XXX
  ```

  and list its contents

  ```
  ls
  ```

- Generate `drawWrightFisher`

  ```
  make
  ```

- List its options

  ```
  ./drawWrightFisher -h
  ```

3 Listing

The following listing documents the driver program for `drawWrightFisher`.

```
/***** drawWrightFisher.c *****************************************
 * Description: Draw tangled and disentangled version of
 *              Wright/Fisher model of evolution.
 * Author: Bernhard Haubold, haubold@evolbio.mpg.de
 * File created on Sat Sep  3 15:38:05 2005.
```

```c
 *
 * This file is part of drawWrightFisher.
 *
 *   drawWrightFisher is free software; you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   drawWrightFisher is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *   GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with drawWrightFisher; if not, write to the Free Software
 *   Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
 ******************************************************************/
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>
#include <gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>
#include "interface.h"
#include "gsl_rng.h"
#include "eprintf.h"
#include "stringUtil.h"
#include "drawWrightFisher.h"

int main(int argc, char *argv[]){
  Args *args;          /* arguments */
  char *version;       /* program version */
  int i, j, k, r, mrcaJ = 0;
  gsl_rng * ranf;
  Cell **pop;
  FILE *fp;
  int *table, *samplePositions, arrayLen;
  char **sampleArray;

  version = "1.3";

  args = getArgs(argc, argv);
  if(args->h == 1){
    printUsage(version);
    return 0;
  }else if(args->e == 1){
    printUsage(version);
    return -1;
  }
  setprogname2("drawWrightFisher");
  ranf = ini_gsl_rng(args);
```

```
/* generate population */
pop = (Cell **)emalloc(args->g*sizeof(Cell *));
for(i=0;i<args->g;i++){
  pop[i] = (Cell *)emalloc(args->p*sizeof(Cell));
  for(j=0;j<args->p;j++){
    pop[i][j].children = (int *)emalloc(args->p*sizeof(int));
    pop[i][j].numDesc = 0;
    pop[i][j].isVisited = 0;
    pop[i][j].generation = i;
    pop[i][j].isAncestral = 0;
    pop[i][j].ancestor = 0;
  }
}
/* generate evolutionary graph */
for(i=1;i<args->g;i++){
  for(j=0;j<args->p;j++){
    r = gsl_rng_uniform(ranf)*args->p;
    pop[i-1][r].children[pop[i-1][r].numDesc] = j;
    pop[i-1][r].numDesc++;
    pop[i][j].ancestor = r;
  }
}
/* untangle graph */
table = (int *)emalloc(args->g*sizeof(int));
for(i=0;i<args->g;i++)
  table[i] = 0;
for(i=0;i<args->g;i++)
  for(j=0;j<args->p;j++)
    if(!pop[i][j].isVisited)
      traverseSubgraph(pop,&pop[i][j],table);

/* mark forward */
if(args->f)
  markForward(pop, &pop[0][args->f-1]);

/* mark ancestral lineages & look for mrca */
sampleArray = (char **)emalloc(args->p*sizeof(char*));
samplePositions = (int *)emalloc((args->p)*sizeof(int));
if(args->a != NULL){
  split(args->a,",",sampleArray,&arrayLen);
  for(i=0;i<args->p;i++)
    samplePositions[i] = 0;
  if(atoi(sampleArray[0]) < 0)
    samplePositions[0] = -1;
  else
    for(i=0;i<arrayLen;i++)
      samplePositions[atoi(sampleArray[i])-1] = 1;
  if(samplePositions[0] == -1)
    for(i=0;i<args->p;i++)
      pop[args->g-1][i].isAncestral = 1;
  else
    for(i=0;i<args->p;i++)
      if(samplePositions[i] == 1)
        pop[args->g-1][i].isAncestral = 1;
  }
/* for(i=args->g-1;i>=0;i--){ */
for(i=args->g-1;i>0;i--){
  k = 0;
  for(j=0;j<args->p;j++)
    if(pop[i][j].isAncestral){
       k++;
       mrcaJ = j;
     }
  if(k==1){
    pop[i][mrcaJ].isMrca = 1;
    break;
  }
  for(j=0;j<args->p;j++)
    if(pop[i][j].isAncestral)
      pop[i-1][pop[i][j].ancestor].isAncestral = 1;
}
fp = efopen(args->o,"w");
/* output tangled version */
fprintf(fp,"\\psset{linewidth=1.5pt}\n");
fprintf(fp,"\\begin{psmatrix}[rowsep=0.2,colsep=0]\n");
fprintf(fp,"\\scalebox{2}{\\textbf{Tangled}}␣&␣\\scalebox{2}{\\textbf{
   Untangled}}\\\\\n");
fprintf(fp,"\\begin{pspicture}(0,0)(%d,%d)\n",args->p+1,-args->g);
i = 0;
fprintf(fp,"\\rput(%d,%d){\\scalebox{1.5}{$g_{%d}$}}",0,-i,i+1);
for(j=0;j<args->p;j++){
  if(args->m && pop[0][j].isMrca)
    fprintf(fp,"\\cnode[fillstyle=solid,fillcolor=red,linecolor=red](%d,%
        d){4pt}{%d%d}",j+1,-i,j,i);
  else if((args->f || args->a ) && pop[i][j].isAncestral)
    fprintf(fp,"\\cnode[linecolor=green,fillstyle=solid,fillcolor=green
        ](%d,%d){4pt}{%d%d}",j+1,-i,j,i);
  else
    fprintf(fp,"\\cnode[fillstyle=solid,fillcolor=black](%d,%d){4pt}{%d%d
        }",j+1,-i,j,i);
}
for(i=1;i<args->g;i++){
  fprintf(fp,"\\rput(%d,%d){\\scalebox{1.5}{$g_{%d}$}}",0,-i,i+1);
  for(j=0;j<args->p;j++){
    if(args->m && pop[i][j].isMrca)
      fprintf(fp,"\\cnode[fillstyle=solid,fillcolor=red,linecolor=red](%d
          ,%d){4pt}{%d%d}",j+1,-i,j,i);
    else if((args->f || args->a ) && pop[i][j].isAncestral)
      fprintf(fp,"\\cnode[linecolor=green,fillstyle=solid,fillcolor=green
         ](%d,%d){4pt}{%d%d}",j+1,-i,j,i);
    else
      fprintf(fp,"\\cnode[fillstyle=solid,fillcolor=black](%d,%d){4pt}{%d
          %d}",j+1,-i,j,i);
  }
  fprintf(fp,"\n");
  for(j=0;j<args->p;j++){
    for(k=0;k<pop[i-1][j].numDesc;k++){
      if((args->a || args->f) && pop[i][pop[i-1][j].children[k]].isAncestral && pop[i-1][j].isAncestral)
        fprintf(fp,"\\ncline[linecolor=green]{%d%d}{%d%d}",j,i-1,pop[i-1][j].children[k],i);
      else
        fprintf(fp,"\\ncline[linestyle=solid]{%d%d}{%d%d}",j,i-1,pop[i-1][j].children[k],i);
    }
  }
  fprintf(fp,"\n");
}
/* label leaf nodes */
for(j=0;j<args->p;j++)
  fprintf(fp,"\\rput(%d,%d){\\scalebox{1.5}{%d}}",j+1,-args->g,j+1);
fprintf(fp,"\n");
fprintf(fp,"\\end{pspicture}\n");
/* output disentangled version */
fprintf(fp,"&\n\\begin{pspicture}(0,0)(%d,%d)\n",args->p+1,-args->g);
i = 0;
for(j=0;j<args->p;j++){
  if(args->m && pop[i][j].isMrca)
    fprintf(fp,"\\cnode[fillstyle=solid,fillcolor=red,linecolor=red](%d,%d){4pt}{%d%d}",pop[0][j].gene+1,-i,pop[0][j].gene,i);
  else if((args->f || args->a ) && pop[i][j].isAncestral)
    fprintf(fp,"\\cnode[linecolor=green,fillstyle=solid,fillcolor=green](%d,%d){4pt}{%d%d}",pop[i][j].gene+1,-i,pop[i][j].gene,i);
  else
    fprintf(fp,"\\cnode[fillstyle=solid,fillcolor=black](%d,%d){4pt}{%d%d}",pop[0][j].gene+1,-i,pop[0][j].gene,i);
}
for(i=1;i<args->g;i++){
  for(j=0;j<args->p;j++){
    if(args->m && pop[i][j].isMrca)
      fprintf(fp,"\\cnode[fillstyle=solid,fillcolor=red,linecolor=red](%d,%d){4pt}{%d%d}",pop[i][j].gene+1,-i,pop[i][j].gene,i);
    else if((args->f || args->a ) && pop[i][j].isAncestral)
      fprintf(fp,"\\cnode[linecolor=green,fillstyle=solid,fillcolor=green](%d,%d){4pt}{%d%d}",pop[i][j].gene+1,-i,pop[i][j].gene,i);
    else
      fprintf(fp,"\\cnode[fillstyle=solid,fillcolor=black](%d,%d){4pt}{%d%d}",pop[i][j].gene+1,-i,pop[i][j].gene,i);
  }
  fprintf(fp,"\n");
  r = 0;
  for(j=0;j<args->p;j++){
    for(k=0;k<pop[i-1][j].numDesc;k++){
      if((args->a || args->f) && pop[i-1][j].isAncestral && pop[i][pop[i-1][j].children[k]].isAncestral)
        fprintf(fp,"\\ncline[linecolor=green]{%d%d}{%d%d}",pop[i-1][j].gene,pop[i-1][j].generation,
                pop[i][pop[i-1][j].children[k]].gene,pop[i][pop[i-1][j].children[k]].generation);
      else
```

```
          fprintf(fp,"\\ncline[linestyle=solid]{%d%d}{%d%d}",pop[i-1][j].
             gene,pop[i-1][j].generation,
                  pop[i][pop[i-1][j].children[k]].gene,pop[i][pop[i-1][j].
                     children[k]].generation);
      }
    }
    fprintf(fp,"\n");
  }
  /* label leaf nodes */
  for(j=0;j<args->p;j++)
    fprintf(fp,"\\rput(%d,%d){\\scalebox{1.5}{%d}}",pop[args->g-1][j].gene
       +1,-args->g,j+1);
  fprintf(fp,"\n");
  fprintf(fp,"\\rput(%d,%d){\\Rnode{Present}{\\psshadowbox{\\scalebox{1.3}{
     Present}}}}\n",j+2,-i+1);
  fprintf(fp,"\\rput(%d,%d){\\Rnode{Past}{\\psshadowbox{\\scalebox{1.3}{
     Past}}}}\n",j+2,0);
  fprintf(fp,"\\ncline{->,linewidth=2pt}{Present}{Past}\n");
  fprintf(fp,"\\end{pspicture}\n");
  fprintf(fp,"\\end{psmatrix}\n");
  fclose(fp);
  printf("Graphic␣written␣to␣%s\n",args->o);
  if(args->t == NULL)
    printf("To␣view␣the␣graphic,␣include␣it␣in␣a␣LaTeX␣file.\n");
  else{
    fp = efopen(args->t,"w");
    printLatexHeader(fp);
    fprintf(fp,"\\begin{center}\\resizebox{\\textwidth}{!}{\\input{");
    i = 0;
    r = strlen(args->o);
    while(args->o[i] != ’.’ && i < r)
      fprintf(fp,"%c",args->o[i++]);
    fprintf(fp,"}}\\end{center}\n\n\\end{document}");
    printf("To␣view␣the␣graphic,␣run␣latex␣%s\n",args->t);
  }
  free_gsl_rng(ranf, args);
  free(table);
  free(samplePositions);
  free(sampleArray);
  /* for(j=0;j<args->p;j++) */
  for(i=0;i<args->g;i++){
    for(j=0;j<args->p;j++){
      free(pop[i][j].children);
    }
    free(pop[i]);
  }
  free(pop);
  free(args);
  free(progname());
  return 0;
}

void printLatexHeader(FILE *fp){
  fprintf(fp,"\\documentclass{article}\n");
  fprintf(fp,"\\usepackage{pstricks,pst-node,graphics,times}\n");
  fprintf(fp,"\\oddsidemargin=0cm\n");
  fprintf(fp,"\\evensidemargin=0cm\n");
  fprintf(fp,"\\pagestyle{empty}\n");
  fprintf(fp,"\\textwidth=16cm\n");
  fprintf(fp,"\\textheight=23cm\n\n");
  fprintf(fp,"\\begin{document}\n");
}


void traverseSubgraph(Cell **pop, Cell *cell, int *table){
  int i;

  cell->gene = table[cell->generation]++;
  cell->isVisited = 1;
  for(i=0;i<cell->numDesc;i++){
    traverseSubgraph(pop,&(pop[cell->generation+1][cell->children[i]]),
       table);
  }

}

void markForward(Cell **pop, Cell *cell){
  int i;
  cell->isAncestral = 1;
  for(i=0;i<cell->numDesc;i++)
    markForward(pop, &(pop[cell->generation+1][cell->children[i]]));
}
```

4 Change Log

- Version 1.2 (February 15, 2017)
  - Fixed missing initiation of the `isAncestral` field.
  - Fixed memory leaks using `valgrind`.
  - Added this documentation.
  - Switched to `gsl_rng` for generating random numbers.
  - Fixed tracing of ancestors.
- Version 1.3 (November 6, 2018)
  - Fixed bug in `interface.c`.