`genTree`, v. 0.5: DESCRIPTION

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

November 6, 2018

1 Introduction

2 Getting Started

`genTree` 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 genTree_XXX.tgz
  ```

  where `XXX` indicates the version.

- Change into the newly created directory

  ```
  cd GenTree_XXX
  ```

  and list its contents

  ```
  ls
  ```

- Generate `genTree`

  ```
  make
  ```

- List its options

  ```
  ./genTree -h
  ```

3 Listing

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

```
/***** genTree.c ***********************************
 * Description: Generate random tree
 * Author: Bernhard Haubold, haubold@evolbio.mpg.de
 * Date: Fri Sep 21 17:01:06 2012
 **************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <string.h>
```

```c
#include <gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>
#include "tree.h"
#include "interface.h"
#include "eprintf.h"
#include "stringUtil.h"

void scanFile(FILE *fp, Args *args);
Node *genTree(Args *args);
void setSpeciationTimes(Node *np);
void setMutations(Node *np);
void printNewickTree(Node *node);

float globalTheta;
gsl_rng *gslGlobal;

int main(int argc, char *argv[]){
  int idum;
  char *version;
  Args *args;
  FILE *fp;
  Node *root;
  const gsl_rng_type *T;

  version = "0.5";
  setprogname2("genTree");
  args = getArgs(argc, argv);
  globalTheta = args->t;
  gsl_rng_env_setup();
  T = gsl_rng_default;
  gslGlobal = gsl_rng_alloc(T);
  /* seed for random number generation */
  if(args->S != 0){
    idum = args->S;
  }else if((fp = fopen("randomSeed.dat","r")) != NULL){
    if(!fscanf(fp,"%d",&idum))
      printf("WARNING[sample.initializeSample]:␣Something␣is␣wrong␣reading␣
          the␣the␣seed␣of␣the␣random␣number␣generator␣from␣randomSeed.dat.\n
          ");
    fclose(fp);
  }else
    idum = -time(NULL);
  gsl_rng_set(gslGlobal,idum);
  if(args->v)
    printSplash(version);
  if(args->h || args->e)
    printUsage(version);
  root = genTree(args);
  if(args->s)
    setSpeciationTimes(root);
  else
    setMutations(root);
  printNewickTree(root);
  free(args);
  free(progname());
  /* save seed of random number generator */
  if(args->S == 0){
    fp = fopen("randomSeed.dat","w");
    fprintf(fp,"%ld\n",gsl_rng_get(gslGlobal));
    fclose(fp);
  }
  gsl_rng_free(gslGlobal);
  return 0;
}

void setSpeciationTimes(Node *np){
  if(np){
    setSpeciationTimes(np->left);
    setSpeciationTimes(np->right);
    if(np->parent)
      np->dist = np->parent->time - np->time;
    else
      np->dist = 0;
  }
}

void setMutations(Node *np){
  float t, mu;
  if(np){
    setMutations(np->left);
    if(np->parent){
      t = np->parent->time - np->time;
      mu = t*globalTheta/2.;
      np->dist = gsl_ran_poisson(gslGlobal,mu);
    }
    setMutations(np->right);
  }
}

Node *genTree(Args *args){
  int i, n, pick, numTaxa;
  Node *p, **tree, *np;
  double t;
  char *buf;

  buf = (char *)emalloc(20*sizeof(char));
  n = args->n;
  numTaxa = 2*n-1;
  tree = (Node **)emalloc(numTaxa*sizeof(Node *));
  for(i=0;i<numTaxa;i++)
    tree[i] = newNode();
  for(i=0;i<n;i++){
    tree[i]->label = emalloc(20*sizeof(char));
    tree[i]->label[0] = ’\0’;
    strcat(tree[i]->label,"T");
    itoa(i+1,buf);
    strcat(tree[i]->label,buf);
  }
  /* generate topology */
  for(i=n;i>1;i--){
    p = tree[2*n-i];
    pick = (int)(gsl_rng_uniform(gslGlobal)*i);
    p->left = tree[pick];
    tree[pick]->parent = p;
    tree[pick] = tree[i-1];
    pick = (int)(gsl_rng_uniform(gslGlobal)*(i-1));
    np = p->left;
    while(np->right)
      np = np->right;
    np->right = tree[pick];
    tree[pick]->parent = p;
    tree[pick] = p;
  }
  /* generate coalescent times */
  for(i=0;i<n;i++)
    tree[i]->time = 0;
  t = 0;
  for(i=n;i>1;i--){
    if(args->c)
      t += -2.*log(1.-gsl_rng_uniform(gslGlobal))/(double)i/(double)(i-1);
    else
      t += -2.*log(1.-gsl_rng_uniform(gslGlobal))/(double)n/(double)(n-1);
    tree[2*n-i]->time = t;
  }
  /* generate mutations */
  for(i=0;i<numTaxa;i++){
    if(tree[i]->parent){
      t = tree[i]->parent->time - tree[i]->time;
      tree[i]->nMut = gsl_ran_poisson(gslGlobal,t*args->t/2.);
    }
  }
  p = tree[numTaxa-1];
  free(tree);
  free(buf);
  return p;
}
```

4 Change Log

- Version 0.1; September 17, 2012
  - First version that worked.
- Version 0.2; January 17, 2012
  - Here is the original code for generating the ancestor times:

    ```
    for(i=n;i>1;i--){
      t += -2.*log(1.-genrand_real1())/(double)n/(double)(i-n);
      tree[2*n-i]->time = t;
    }
    ```

    This returns tree that look remarkably like phylogenies, where in many cases the branches near the root are short. What I *meant* to program were the standard coalescent times

```
t += -2.*log(1.-genrand_real1())/(double)i/(double)(i-1);
```

However, these give visually less convincing phylogenies and hence this computation is only used in response to option `-c`.

- Version 0.3; December 19, 2014
  - Compute topology before times. Not sure why I did this any more, but still moving on with this version.
- Version 0.4; June 3, 2015
  - Cleaned random number generation.
- Version 0.5; November 6, 2018
  - Fixed bug in `interface.c`.