rush, v. 1.4: Recombination detection Using SHustrings

Bernhard Haubold[1], Linda Krause[1,2], Thomas Horn[3], Peter Pfaffelhuber[3]

[1]Evolutionary Genetics, Max-Planck-Institute for Evolutionary Biology, Plön, Germany

[2]Lübeck University, Lübeck, Germany

[3]Mathematical Stochastics, Mathematical Institute, Albert-Ludwigs University, Freiburg, Germany

August 15, 2013

1 Introduction

Recombination is traditionally thought to speed up adaptation [2, 6] and to eliminate deleterious mutations from populations [7]. It is therefore a central mechanism of evolution and has been studied extensively by theoreticians and experimentalists alike [4]. `rush`, which stands for "Recombination detection Using SHustrings", is a program for determining whether or not recombination has taken place during the evolution of a pair of homologous DNA sequences. There are already many programs available for doing this [8, 1]. The unique feature of `rush` is that it analyzes unaligned genomes. This makes the program very fast.

2 Getting Started

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

- Unpack the program

  ```
  tar -xvzf rush_XXX.tgz
  ```

  where `XXX` indicates the version.

- Change into the newly created directory

  ```
  cd Rush_XXX
  ```

  and list its contents

  ```
  ls
  ```

- Generate `rush`

  ```
  make
  ```

- List its options

  ```
  ./rush -h
  ```

3 Tutorial

- First install the following additional programs

Program	Author	Source	Reference
`ms`	R. R. Hudson	`home.uchicago.edu/rhudson1/`	[5]
`ms2dna`	B. Haubold & P. Pfaffelhuber	`guanine.evolbio.mpg.de/bioBox/`	—
`getSeq`	B. Haubold	`guanine.evolbio.mpg.de/bioBox/`	—

and then execute from within the directory `Rush_XXX`

```
sh Scripts/test.sh
```

where `test.sh` is

```
# The following command breaks down as
#   ms 2 1: run Hudson's ms to generate 1 set of 2 haplotypes
#   -s 1000: the sequences are separated by 1000 mutations
#   -r 100 100000: the sequences undergo an expected 100
#      recombination events and are 100000 bp long
ms 2 1 -s 1000 -r 100 100000 |
# Convert the sequences to DNA
ms2dna > tmp.fasta;
# Extract query & subject sequences
getSeq -s S1 tmp.fasta > query.fasta;
getSeq -s S2 tmp.fasta > sbjct.fasta;
# Run rush
./rush -q query.fasta sbjct.fasta
```

An example result is

```
Q 2.611e+00 D_r 5.426e+00 P 1.550e-08
```

Yours will differ, because `test.sh` generates random test sequences. In the result, $Q = 2.611$ is our recombination measure, $D_{\mathrm{r}} = 5.426$ the test statistic, and $P = 1.55 \times 10^{-8}$ the error probability when rejecting $H_0 : D_{\mathrm{r}} = 0$.

- Next, explore the rejection frequency as a function of the rate of recombination:

```
awk -f Scripts/rush.awk
```

where `rush.awk` is

```
BEGIN{
  template = "ms 2 1 -s 1000 -r %f 100000 | ms2dna > tmp.fasta; getSeq
      -s S1 tmp.fasta > s1.fasta; getSeq -s S2 tmp.fasta > s2.fasta;
     rush -q s1.fasta s2.fasta | grep -v '^s'";
  minRho = 1;
  maxRho = 4096;
  it = 10;
  print "#rho\tRejection (alpha = 0.05)"
  for(r=minRho;r<=maxRho;r*=2){
    cmd = sprintf(template,r);
    c = 0;
    for(j=0;j<it;j++){
```

```
11        cmd | getline;
          if($6 <= 0.05)
            c++;
          close(cmd);
        }
16      print r "\t" mean "\t" c/it;
      }
    }
```

Your results should look similar to

```
#rho Rejection (alpha = 0.05)
1 0
2 0.2
4 0.2
8 0.4
16 0.8
32 0.8
64 0.8
128 1
256 1
512 1
1024 1
2048 0.9
4096 1
```

Notice that you can change the number of iterations in `rush.awk` from 10 to some larger value, say 100, by changing line 5 from

```
it = 10;
```

to

```
it = 100;
```

4 Listings

4.1 Driver Program `rush.c`

```
/***** rush.c ************************************
2  * Description: Recombination detection Using
   *   SHustrings
   * Author: Bernhard Haubold, haubold@evolbio.mpg.de
   * Date: Thu Apr  4 17:17:34 2013
   ***********************************************/
7  #include <stdio.h>
   #include <stdlib.h>
   #include <unistd.h>
   #include <fcntl.h>
   #include <math.h>
12 #include <gsl/gsl_sf_gamma.h>
   #include <gsl/gsl_errno.h>
   #include "DeepShallow64/common.h"
   #include "interface.h"
```

```
#include "eprintf.h"
#include "sequenceData.h"
#include "lcpTree.h"
#include "minimize.h"
#include "rush.h"
#include "prob.h"

void scanFile(int fd, Args *args);
double pid(Int64 *sl, Int64 len, double gc, int min);
int  minShulen(int sbjctLen, double gc, double threshold);
double absolute(double a);

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

  version = "1.4";
  setprogname2("rush");
  args = getArgs(argc, argv);
  if(args->v)
    printSplash(version);
  if(args->h || args->e)
    printUsage(version);
  gsl_set_error_handler_off();
  if(args->numInputFiles == 0){
    sbjctDscr= 0;
    scanFile(sbjctDscr, args);
  }else{
    for(i=0;i<args->numInputFiles;i++){
      sbjctDscr = open(args->inputFiles[i],0);
      scanFile(sbjctDscr, args);
      close(sbjctDscr);
    }
  }
  free(args);
  free(progname());
  return 0;
}

void scanFile(int sbjctDscr, Args *args){
  Int64 *sl, i, len;
  double s, meanSl, sx, sig, ev, varSl, q, dr;
  Sequence *query, *sbjct, *seq;
  int queryDscr, r, l, winLen;

  queryDscr = open(args->q,0);
  if(queryDscr < 0)
    eprintf("ERROR:␣could␣not␣open␣query␣file␣%s\n",args->q);
  query = readFasta(queryDscr);
  sbjct = readFasta(sbjctDscr);
  close(sbjctDscr);
  prepareSeq(query);
  prepareSeq(sbjct);
```

```
    len = query->len/2 - 1;
    seq = catSeq(query,sbjct);
72  freeSequence(query);
    freeSequence(sbjct);
    sl = getLcpTreeShulens(args, seq);
    s = 0.0;
    sx = 0.0;
77  l = 0;
    r = 0;
    if(!args->w){
      winLen = len;
    }else{
82    winLen = args->w;
    }
    /* fill first window */
    for(r=0;r<winLen;r++){
      s += sl[r];
87    sx += sl[r]*sl[r];
    }
    meanSl = s/winLen;
    varSl = (sx-s*s/winLen)/(winLen-1);
    ev = eVar(meanSl,winLen);
92  sig = significanceVar(meanSl, varSl, winLen);
    dr = (varSl - ev)/sqrt(24.0*pow(meanSl,5)/winLen);
    q = varSl/ev;
    if(args->w)
      printf("Pos\t%d\tQ\t%.3e\tD_r\t%.3e",(int)((r+l)/2.),q,dr);
97  else
      printf("Q\t%.3e\tD_r\t%.3e",q,dr);
    if(sig != -1){
      if(dr > 0.0)
        printf("\tP\t%.3e\n",sig);
102   else
        if(sig > 0.05)
          printf("\tP\t%.3e\n",sig);
        else
          printf("\tP\tfailed\n");
107 }else
      printf("\tP\tfailed\n");
    /* scan remaining windows */
    while(r<len-args->s){
      for(i=0;i<args->s;i++){
112     s += sl[r];
        s -= sl[l];
        sx += sl[r]*sl[r];
        sx -= sl[l]*sl[l];
        l++;
117     r++;
      }
      meanSl = s/winLen;
      varSl = (sx-s*s/winLen)/(winLen-1);
      ev = eVar(meanSl,winLen);
122   sig = significanceVar(meanSl, varSl, winLen);
      dr = (varSl - ev)/sqrt(24.0*pow(meanSl,5)/winLen);
```

```
      q = varSl/ev;
      printf("Pos\t%d\tQ\t%.3e\tD_r\t%3.e",(int)((r+l)/2.),q,dr);
      if(sig != -1){
        if(dr > 0.0)
          printf("\tP\t%.3e\n",sig);
        else
          if(sig > 0.05)
            printf("\tP\t%.3e\n",sig);
          else
            printf("\tP\tfailed\n");
      }else
        printf("\tP\tfailed\n");
    }
    freeSequence(seq);
    free(sl);
  }
```

4.2 Variance Computation `varSd.c`

```
  /***** varSd.c ************************************
   * Description:
   * Author: Bernhard Haubold, haubold@evolbio.mpg.de
   * Date: Thu Jan 10 12:39:59 2013
   ******************************************************/
  #include <stdio.h>
  #include <math.h>
  #include <gsl/gsl_sf_erf.h>
  #include <gsl/gsl_cdf.h>
  #include <gsl/gsl_randist.h>
  #include "DeepShallow64/common.h"
  #include "rush.h"

  double expVarVarSl(double l, double p){
    double vvs;
    double p5, p6, p7, p8;
    double l2, l3, l4;

    p5 = p*p*p*p*p;
    p6 = p5*p;
    p7 = p6*p;
    p8 = p7*p;

    l2 = l*l;
    l3 = l2*l;
    l4 = l3*l;

    vvs = 24./l/p5 - 264/l2/p6 + 1320/l3/p7 - 2778/l4/p8;

    return vvs;
  }

  double significanceVar(double meanSl, double varSl, int winLen){
    double p, x, sig, evvs;

    p = 1/meanSl;
```

```
  evvs = expVarVarSl(winLen,p);
  if(evvs > 0){
    x = (varSl-eVar(meanSl,winLen))/sqrt(evvs);
    sig = gsl_cdf_ugaussian_Q(x);
    if(sig > 1)
      return 1.0;
    else
      return sig;
  }else
    return 0.0;
}

double eVar(double meanSl, double l){
  double p, vs;

  p = 1./meanSl;
  vs = 1./p/p - 2./l/p/p/p + 2./l/l/p/p/p/p;

  return vs;
}
```

5 Change Log

- Version 0.1 (July 31, 2012)
  - Initial version.
- Version 0.2 (August 3, 2012)
  - Cut out all superfluous code inherited from the computation of π_{d}; this made the program much faster.
- Version 0.3 (August 25, 2012)
  - Implemented the option to include only shustrings of a minimum length (`-m`).
- Version 0.4 (September 10, 2012)
  - Implemented Peter Pfaffelhuber's new formula for pi_{d} to estimate π if not supplied by user.
- Version 0.5 (November 15, 2012)
  - Included the fast version of the full pi_{d} computation. The minimum shustring length now becomes important.
- Version 0.6 (November 16, 2012)
  - Adjusted computation of minimum shustring length to accommodate GC-contents $\neq 1/2$.
- Version 0.7 (November 17, 2012)
  - Exact computation of minimum shustring length; addition of `-t` option for determining the fraction of random shustring lengths ignored.
- Version 0.8 (November 19, 2012)
  - Mixed estimation of minimum shustring length: if the GC-content deviates from 0.5 by more than `-g`, explicit computation of minimum shustring length is used; otherwise Section 4 of the Memo dated August 28 2012 by Haubold, Horn, & Pfaffelhuber is used.
- Version 0.9 (November 23, 2012)

  - Fraction of rejected shustring lengths computed as a function of sequence length.
- Version 0.10 (November 29, 2012)
  - Implemented the new R-statistic.
- Version 0.11 (December 6, 2012)
  - Sliding window analysis of R.
- Version 0.12 (December 14, 2012)
  - Fixed error in sliding window analysis.
- Version 0.13 (???)
- Version 0.14 (January 28, 2013)
  - Implemented new hypothesis test.
  - Included significance computation.
- Version 0.15 (January 29, 2013)
  - Abolished R.
  - Used $\overline{X}^2 = (1 - \pi)/\pi^2$.
  - Caught negative expected variance in `varSd.significanceVar`.
- Version 0.16 (February 1, 2013)
  - Implemented extreme value distribution for hypothesis testing.
- Version 0.17 (February 4, 2013)
  - Set negative expected variance to 0 in `varSd.significanceVar`.g
- Version 0.18 (February 8, 2013)
  - Implemented explicit error handling [3, p. 18f].
- Version 0.19 (February 12, 2013)
  - Reverted to Gaussian null distribution.
  - Implemented switch to test with log-transformed data.
- Version 0.20 (March 8, 2013)
  - Removed memory leak in `lcpTree.c` by replacing the increment of `maxNumLeaves` and `maxNumChildren` by `++` instead of `*2`.
  - Removed option for log-transformation.
- Version 0.21 (April 10, 2013)
  - Switched from `recTest` to `rush`.
- Version 1.0 (April 12, 2013)
  - First version released on web site.
- Version 1.1 (April 19, 2013)
  - Implemented Peter's simplified expressions for $E[s^2]$ and $Var[s^2]$.
- Version 1.2 (May 13, 2013)

  - Changed nomenclature in output from `V_o` and `V_e` to `s^2` and `X^2`.

- Version 1.3 (May 17, 2013)
  - There was an error in the significance computation. Changed in `varS.c`

    ```
    sig = gsl_sf_erfc(x);
    ```

    to

    ```
    sig = gsl_cdf_ugaussian_Q(x);
    ```

    which is equivalent to

    ```
    sig =  gsl_sf_erfc(x/sqrt(2.))/2.;
    ```

  - Asked the user for a query file in response to the `-v` and `-h` options. Fixed.

- Version 1.4 (August 15, 2013)
  - Changed the output to Q, $D_{\rm r}$, and its significance.
  - If $D_{\rm r} < 0$, don't report significance values < 0.05. Unfortunately, if $D_{\rm r}$ is only a bit greater than 0, `rush` can report $P = 0$ if the sequences are very short (e.g. 1 kb):

    ```
    for a in $(seq 100)
       do generateQuerySbjct -l 1000 -s 10 -r 45
       ./rush -q query.fasta sbjct.fasta
    done | grep 0.000
    ```

    I don't know how to fix this, so I am leaving it as it is for now.

-

References

[1] T. C. Bruen, H. Philippe, and D. Bryant. A simple and robust statistical test for detecting the presence of recombination. *Genetics*, 172:2665–2681, 2006.

[2] R. A. Fisher. *The Genetical Theory of Natural Selection*. Oxford University Press, Oxford, Variorum edition, 1930/1999.

[3] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi. *GNU Scientific Library Reference Manual*. Network Theory Ltd, 1.6, for gsl version 1.6, 17 march 2005 edition, 2005.

[4] W. D. Hamilton. *Narrow Roads of Gene Land*, volume 2. Oxford University Press, 2001.

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

[6] H. J. Muller. Some genetic aspects of sex. *American Naturalist*, 66:118–138, 1932.

[7] H. J. Muller. The relation of recombination to mutational advance. *Mutation Research*, 1:2–9, 1964.

[8] D. Posada. Evaluation of methods for detecting recombination from DNA sequences: empirical data. *Molecular Biology and Evolution*, 19:708–717, 2002.