`cutSeq`, v. 0.14: Cut Regions from Molecular Sequences

Bernhard Haubold

November 6, 2018

1 Introduction

`cutSeq` is a computer program for cutting regions from one or more FASTA formatted sequences. In this document I describe how to get started with `cutSeq` and then demonstrate its usage in a brief Tutorial.

2 Getting Started

`cutSeq` was written in C on a computer running Mac OS X; it is intended to run on any UNIX system with a C compiler. However, please contact me at `haubold@evolbio.mpg.de` if you have problems with the program.

- Unpack the program

  ```
  tar -xvzf cutSeq_XXX.tgz
  ```

  where `XXX` indicates the version.
- Change into the newly created directory

  ```
  cd CutSeq_XXX
  ```

  and list its contents

  ```
  ls
  ```
- Generate `cutSeq`

  ```
  make
  ```
- List its options

  ```
  ./cutSeq -h
  ```
- Test the program

  ```
  cutSeq -r 501-520 test.fasta
  ```

3 Tutorial

- Use a comma-separated list to cut out more than a single region

  ```
  ./cutSeq -r 501-520,531-550 test.fasta
  >S1 501..520
  CTAGAGAAAGGTTTGGGAGC
  >S1 531..550
  TGGTGACCTTGTGGGCATGC
  ```

- Region coordinates can also be passed to `cutSeq` via a file. This is may be handy when extracting a large number of regions.

  ```
  ./cutSeq -f regions.txt test.fasta
  >S1 501..520
  CTAGAGAAAGGTTTGGGAGC
  >S1 531..550
  TGGTGACCTTGTGGGCATGC
  ```

- Sometimes we wish to splice the regions

  ```
  ./cutSeq -s -f regions.txt test.fasta
  >S1 join(501..520,531..550)
  CTAGAGAAAGGTTTGGGAGCTGGTGACCTTGTGGGCATGC
  ```

4 Change Log

1. v. 0.11 (February 9, 2011)
    - First version distributed.
2. v. 0.12 (March 7, 2017)
    - Fixed small bug in interface.
3. v. 0.13 (March 22, 2017)
    - Fixed severe bug in interface.
4. v. 0.14 (November 6, 2018)
    - Fixed bug in `interface.c`.

5 Listing

5.1 The Driver Program: `cutSeq.c`

```
/***** cutSeq.c *************************************************
 * Description: Cut portions from molecular sequences.
 * Author: Bernhard Haubold, haubold@evolbio.mpg.de
 * File created on Sun Mar 27 18:22:03 2005.
 *****************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <time.h>
#include <assert.h>
#include <fcntl.h>
#include "eprintf.h"
#include "stringUtil.h"
#include "interface.h"
#include "sequenceData.h"
#include "cutSeq.h"
```

```c
void runAnalysis(Args *args, int fd, FILE *fpf, Interval *intervals, int
    arrayLen);

int main(int argc, char *argv[]){
  Args *args;           /* arguments */
  char *version;        /* program version */
  int fd;
  int i;
  FILE *fpf;
  char **positionArray, **positionPair;
  int arrayLen, pairLen;
  Interval *intervals;

  version = "0.14";
  args = getArgs(argc, argv);
  setprogname2("cutSeq");

  positionArray = positionPair = NULL;
  fpf = NULL;
  if(args->h == 1){
    printUsage(version);
    return 0;
  }else if(args->v == 1){
    printSplash(version);
    return 0;
  }else if(args->e == 1){
    printUsage(version);
    return -1;
  }
  /* compute start and endpositions of regions */
  positionArray = (char **)emalloc(MAXFIELDS*sizeof(char));
  positionPair = (char **)emalloc(2*sizeof(char));
  if(args->r){
    split(args->r,",",positionArray,&arrayLen);
    intervals = (Interval *)emalloc(arrayLen * sizeof(Interval));
    for(i=0;i<arrayLen;i++){
      split(positionArray[i],"-",positionPair,&pairLen);
      intervals[i].start = atoi(positionPair[0])-1;
      intervals[i].end = atoi(positionPair[1])-1;
    }
  }else if(args->f){
    fpf = efopen(args->f,"r");
    arrayLen = 0;
    intervals = (Interval *)emalloc(sizeof(Interval));
    while(fscanf(fpf,"%d␣%d",&intervals[arrayLen].start,&intervals[arrayLen
        ].end) != EOF){
      intervals[arrayLen].start--;
      intervals[arrayLen].end--;
      arrayLen++;
      intervals = (Interval *)erealloc(intervals,(arrayLen+1)*sizeof(
          Interval));
    }
    fclose(fpf);
  }else
```

```
    exit(0);

  if(args->numInputFiles == 0){
    runAnalysis(args,0,fpf,intervals,arrayLen);
  }else{
    for(i=0;i<args->numInputFiles;i++){
      fd = open(args->inputFiles[i],0);
      if(fd < 0){
        printf("ERROR␣[cutSeq]:␣could␣not␣open␣input␣file␣%s\n",args->inputFiles[i]);
        return -1;
      }
      runAnalysis(args,fd,fpf,intervals,arrayLen);
    }
  }

  return 0;
}

void runAnalysis(Args *args, int fd, FILE *fpf, Interval *intervals, int arrayLen){
  int i, j, k;
  int tmp;
  Sequence *subject, **seqArray, *seq;
  char c, *spliced;
  int pos, end, start, seqLen;

  subject = NULL;
  seq = readFasta(fd);
  seqArray = sequence2array(seq);
  for(i=0;i<seq->numSeq;i++){
    subject = seqArray[i];
    seqLen = strlen(subject->seq) - 1;
    if(args->s){                                  /* splicing */
      spliced = emalloc((seqLen+1)*sizeof(char));
      qsort(intervals, arrayLen, sizeof(intervals[0]), icmp);
      fprintf(stdout,"%s␣join(",subject->headers[0]);
      pos = 0;
      for(j=0;j<arrayLen;j++){
        if(j)
          fprintf(stdout,",");
        if(intervals[j].end < intervals[j].start){
          tmp = intervals[i].end;
          intervals[j].end = intervals[j].start;
          intervals[j].start = tmp;
        }
        fprintf(stdout,"%d..",intervals[j].start+1);
        fprintf(stdout,"%d",intervals[j].end+1);
        start = intervals[j].start;
        end = (intervals[j].end < seqLen - 1) ? intervals[j].end : seqLen - 1;
        if(start>end)
          continue;
        for(k=start;k<=end;k++)
```

```c
          spliced[pos++] = subject->seq[k];
      }
      fprintf(stdout,")\n");
      spliced[pos] = ’\0’;
      printWrap(stdout,spliced,args->l);
      free(spliced);
    }else{                                   /* cutting */
      for(j=0;j<arrayLen;j++){
        start = intervals[j].start;
        end = (intervals[j].end < seqLen - 1) ? intervals[j].end : seqLen - 1;
        if(start>end)
          continue;
        fprintf(stdout,"%s␣%d..%d\n",subject->headers[0],start+1,end+1);
        c = subject->seq[end+1];
        subject->seq[end+1] = ’\0’;
        printWrap(stdout,subject->seq+start,args->l);
        subject->seq[end+1] = c;
      }
    }
  }
  for(i=0;i<seq->numSeq;i++)
    freeSequence(seqArray[i]);
  free(seqArray);
  freeSequence(seq);

}

int icmp(const void *p1, const void *p2){
  Interval interval1, interval2;

  interval1 = *(Interval *)p1;
  interval2 = *(Interval *)p2;
  if(interval1.start < interval2.start)
    return -1;
  else if(interval1.start > interval2.start)
    return 1;
  else
    return 0;
}
```