

ir Version 2.3: Program for Calculating the Index of Repetitiveness, I_r , from DNA Sequences

Bernhard Haubold and Thomas Wiehe

May 20, 2007

DNA sequences vary strongly in their repetitiveness. However, the biological causes and consequences of this repetitiveness remain unclear. To put the investigation of DNA repetitiveness on a firm quantitative footing, we have devised the index of repetitiveness, I_r , which is ≈ 0 for sequences that contain only the amount of repeats expected under randomness, and > 0 for sequences with an excess of repeats [1].

Downloading and Compiling ir

A program implementing sequence analysis based on the I_r is freely available under the GNU General Public License. The following instructions for using this software are based on a computer running the LINUX operating system. However, with a bit of tweaking they should also be applicable to other platforms.

Unpack ir

```
tar -xvzf ir_XXX.tgz
```

and change into the resulting directory

```
cd Ir_XXX
```

Compile the program by typing

```
make
```

The command

```
./ir -h
```

should now print the following list of program options:

```
ir version 2.3, copyright (c) 2006-2007 Bernhard Haubold & Thomas Wiehe
distributed under the GNU General Public License.
```

```
purpose: calculate the index of repetitiveness,  $I_r$ 
```

```
usage: ir [options]
```

```
options:
```

```
[-i <FILE> read input from FILE; default: FILE=stdin]
[-o <FILE> write output to FILE; default: FILE=stdout]
[-w <NUM> sliding window of width NUM; default: no sliding window]
[-c <NUM> increment sliding window by NUM positions; default: window_width/10]
[-n <NUM> print NUM characters of header; all if NUM<0; default: NUM=30]
[-s treat each sequence separately; default: union]
[-p print information about program]
[-h print this help message]
```

Data Format

`ir` accepts FASTA formatted input data. The program only recognizes positions occupied by one of the four characters A, C, G, and T, which can be either in upper or lower case. Other characters are treated as missing data. In the computation of global I_r they are ignored. In the window analysis only windows consisting entirely of the four canonical characters are considered. Windows containing other sequence data are assigned an I_r value of “n/a” for “not available”.

Tutorial

Global I_r

To test `ir`, download the example genome of *Mycoplasma genitalium* into the newly created directory `Ir_xxx` and uncompress it:

```
gunzip 143967.fasta.gz
```

To calculate the I_r for this sequence, enter

```
./ir -i 143967.fasta
```

to get

```
# Len    I_r
580076   0.1388
```

We can compare this to the I_r value of the shuffled version of the genome by typing

```
shuffleseq -filter 143967.fasta | ./ir
```

where `shuffleseq` is part of the EMBOSS software package [3]. The I_r of the shuffled sequence should be close to zero.

Window Analysis

Instead of just computing an I_r for an entire sequence, it is often more informative to look at local variations in I_r . This can be done by carrying out a window analysis using the `-w` option. In order to visualize the result of such a window analysis, we use the program `graph`, which is part of the `plotutils` package. Given a functioning installation of `plotutils`, try

```
./ir -w 1000 < 143967.fasta \  
| graph -y 0 -X Ir -Y Position -T X
```

where `\` indicates line continuation. Notice the very sharp peaks in the resulting plot corresponding to regions with exceptional repetitiveness. We can smooth the plot by increasing the window size to, say, 10,000:

```
./ir -w 10000 < 143967.fasta \  
| graph -y 0 -X Ir -Y Position -T X
```

In order to print our plot, generate it first in `xfig` format, for example:

```
./ir -w 10000 < 143967.fasta \  
| graph -y 0 -X Ir -Y Position -T fig > mg.fig
```

The resulting file can now be manipulated using the program `xfig`. Alternatively, we can convert it to postscript by typing

```
fig2dev -L ps mg.fig > mg.ps
```

Similarly, encapsulated postscript for inclusion in, e.g., \LaTeX documents can be generated by

```
fig2dev -L eps mg.fig > mg.eps
```

Multiple Sequences

Multiple sequences can be analyzed either separately, or combined. To see the difference between these two modes, compute the I_r of the random sequence supplied together with the source files:

```
cat ranSeq.fasta | ./ir # Len    I_r
10000    0.0012
```

If we add another exact copy of this sequence to the analysis, we expect to see a great increase in I_r :

```
cat ranSeq.fasta ranSeq.fasta | ./ir
# Len    I_r
20000    6.3702
```

However, if we switch on the separate mode of analysis, we return to the low I_r value of the original isolated sequence:

```
cat ranSeq.fasta ranSeq.fasta | ./ir -s
# Seq    Len    I_r
>Random Sequence #1; G/C=0.50    10000    0.0012
>Random Sequence #1; G/C=0.50    10000    0.0012
```

When carrying out a window analysis on multiple sequences it might be useful to write the I_r values of the different sequences into distinct files. This can be done using the UNIX tool `csplit`. For our biologically rather meaningless example we can execute

```
cat ranSeq.fasta ranSeq.fasta | ./ir -w 100 | csplit -f dataset -z - /\>/ {1}
```

to find the I_r values for the first sequence in `dataset01` and those for the second sequence in `dataset02`.

Change Log

1. Up to version 1.1: based on suffix tree as described in our original publication [1].
2. Since version 2.0: based on suffix array, which can be thought of as a space-efficient representation of a suffix tree. Also gained in speed.
3. Version 2.2: switched from locally varying GC content in sliding window analysis to constant CG content. In other words, the expected aggregate shustring length, A_e (cf. [1]) is the same for all windows.
4. Version 2.3 (May 20,2007): improved user interface.

Acknowledgments

`ir` is based on the `dss_sort` library by G. Manzini [2]. We are grateful to Johannes Fischer for drawing our attention to this software library and to Linda Serpling for helpful comments on an earlier version of `ir`.

References

- [1] B. Haubold and T. Wiehe. How repetitive are genomes? *BMC Bioinformatics*, 7:541, 2006.
- [2] G. Manzini and P. Ferragina. *Engineering a lightweight suffix array construction algorithm*, pages 698–710. Number 2461 in Springer Verlag Lecture Notes in Computer Science. Springer Verlag, 2002.
- [3] P. Rice and A. Bleasby. EMBOSS: The European Molecular Biology Open Software Suite. *Trends in Genetics*, 16:276–277, 2000.