GlbPSs 1.0 Manual Part 6: Documentation of poploc_09.0.pl

5/22/2015 Andreas Hapke,
Institute of Anthropology, Johannes Gutenberg University Mainz, Germany, ahapke2@gmail.com

This program identifies loci and alleles at the level of the total study population, i.e. all individuals that you analyzed with **indloc**. After locus identification, it searches for overlapping fragments in allele sequences and merges them.

Algorithm overview

The program analyzes the sequences of all alleles that **indloc** determined in all individuals. It performs pairwise comparisons between allele sequences of the same length and identifies pairs with a distance <= *interind_d*. You can specify different values of *interind_d* for different sequence lengths. The program clusters alleles into networks wherein each allele has a distance <= *interind_d* to at least one other allele. Each network is a population locus. The nodes are all known alleles of the locus. Alleles that are not part of any pair constitute separate population loci. The program then produces a catalog of population loci with the sequences of their alleles and further data.

The algorithm assembles individual allele sequences into population loci based on two sources of information: the allele sequences themselves and *interind_d*. During locus identification, it does not consider which alleles belong to the same individual locus in specific individuals. It can thus happen that a population locus comprises alleles from several individual loci from the same individual. These alleles have been tied together into one network by **poploc**, but not by **indloc**. How often this happens is influenced by the the distance settings in **indloc** (*good_svar_d* and *rare_svar_d*) and **poploc** (*interind_d*) and by the variability of a locus within and between individuals. The program identifies such loci after locus identification and flags them with value 1 in colum “ties” in outfile `poploc.txt`.

Likewise, it can happen that **poploc** assigns two alleles to different population loci although **indloc** had assigned them to the same individual locus in one or several individuals. The next program, **indpoploc**, automatically identifies such split loci.

You can later select or unselect tied loci and split loci with the aid of the program **data_selector**. Furthermore, the programs **pair_finder** and **depth_analyzer** automatically ignore split loci.

Merging of overlapping fragments in allele sequences

This function is useful when you have constructed *rc-duplicates* with **fdm** before the analysis with **indloc**. Merging is necessary for sequences that did not contain adapters. **fdm** constructs *rc-duplicates* from read pairs without adapter as follows: It truncates the longer read to the length of the shorter read. (The shorter read is the one that contained the barcode, which is already removed now.) **fdm** then constructs an *rc-duplicate* pair by concatenating the f-read with the reverse complement of the r-read and building the reverse complement of this concatenated read. The resulting sequences consist of two fragments of equal length that may or may not overlap. All of these sequences have the same length. They are longer than all sequences that contained adapters unless you used *f_tr* and *r_tr* in **fdm** to truncate them to an extremely short length, which you did not, as I suppose. You can instruct **poploc** to merge overlapping fragments in sequences of one specific length. The algorithm analyzes allele sequences separately for each population locus: It splits each allele into two fragments of equal length and identifies all possible overlaps between them without mismatch. It then merges all alleles of the locus using the longest overlap that is possible for all alleles. Repeated sequence motifs or genotyping errors can cause a situation where the longest possible overlap differs between the alleles of a locus. The program flags these loci in the outfile `poploc.txt`. You can later remove them with the aid of the program **data_selector**.

Usage

```
poploc_09.0.pl
```
or
```
perl poploc_09.0.pl
```

Several command flags enable you to control the analysis. Example:

```
poploc_09.0.pl -d distance_file.txt -msl 174 -mino 10 -spl 90
```

Command flags

-d	Name (or path) of a file with distance settings.
-msl	Merge overlapping fragments in loci of specified length. Use `-msl 0` to inactivate merging.
-mino	Minimum overlap required for merging.
-spl	Shortest plausible locus length after merging.

Command flags in detail

-d Name (or path) of a file with distance settings

The file must have this format: textfile two non-negative integers per line, separated by TABs: sequence length and *interind_d*

It is not necessary to define *interind_d* for all occurring sequence lengths: Example:

```
30	6

60	8
```

With these distance settings, poploc will use *interind_d* 6 for all sequences with lengths from 1 to 59 and *interind_d* 8 for all sequences with length 60 and greater.

Default: The program will use *interind_d* = 6 for all sequence lengths when it cannot open the file or when you do not provide a filename.

Command flags for merging of overlapping fragments: -msl -mino -spl

I use an example here to explain these flags: You have used **fdm** to analyze paired reads and to construct *rc-duplicates.* The reads have a length of 100. Your f-reads contained the barcode and the longest barcode had length 8. You have activated `compbcl` to shorten all f-reads to the same length after barcode removal. You have used BamH1 (`G^GATCC`). Sequences of your loci begin with `GATCC` and end with `GGATC`. You have searched for adapters in both reads. Adapter sequences begin with `GATC`. You have used `f_ao -1` and `r_ao -1` to completely remove the restriction site from adapter containing sequences. You have also removed the restriction site from sequences without adapter. fdm needs 5 bases of the adapter at the end of a sequence to detect an adapter. The longest *rc-duplicates* from sequences with adapter will have a length of 89. (100 minus 6 bases at the end and 5 bases at the beginning of the r-read). All *rc-duplicates* from sequences without adapter will have a length of 174: The f-read has length 92 after removal of the barcode and length 87 after removal of the restriction site. fdm truncates the r-read to the same length and appends its reverse complement to the f-read.

-msl Merge overlapping fragments in loci of specified length

With the example above, we use `-msl 174` to merge overlapping fragments. Use `-msl 0` to inactivate merging. The value of -msl must be 0 or an even positive integer. If not, the program reverts to the default setting: `-msl 0`.

-mino Minimum overlap required for merging

Use this flag to set the minimum number of overlapping positions required for merging. The default is 10. The value of mino must be a positive integer ≤ msl/2. If not, the program reverts to the default setting: 10 or msl/2 if msl < 20.

-spl Shortest plausible locus length after merging

Per default, the algorithm always uses the longest overlap possible for all alleles of a locus when merging overlapping fragments. When a locus contains a repeat motif, there can be several possible overlaps and the longest one may not be the true one. At least you can set a shortest plausible locus length after merging. In our example above, the longest *rc-duplicates* from sequences with adapter have a length of 89. You should then use `-spl 90`. The reason is that **fdm** would have detected adapters in reads of a locus shorter than 90. It would thus not have concatenated the f and r read to construct *rc-duplicates*. Use `-spl 100` if, for any reason, you did not remove the restriction site from sequences without adapter and everything else corresponds to the example above. The value of spl must be an integer ≥ msl/2. If not, the program reverts to the default setting spl = msl/2.

Infiles

Apart from the distance settings file explained above, all infiles for **poploc** are outfiles of **indloc**: the file `individuals.txt` and, from each individual, the file `ind_ID_alleles.txt` (where “ind_ID” is the ID of an individual).

Outfiles

poploc produces three outfiles. If any of these outfiles already exists in the directory where **poploc** is, it overwrites it.

`poploc_report.txt`

This file contains the settings used by the program, the IDs of the analyzed individuals, the runtime and error messages (if any).

`poploc.txt`

Format: tab-delimited text table with header line. One line per population locus:

poplocID: ID of this locus

sl: sequence length

cons: consensus sequence of all known alleles with IUPAC ambiguity symbols at variable positions

nSNP: number of variable positions

varpos: variable positions, position count starting with 1, format: p:4/5/12 means three variable positions: 4,5 and 12.

n_all: number of alleles

nInd: number of individuals that have contributed alleles to the population locus

nIndloc: number of individual loci that have contributed alleles to the population locus

ties: 0/1: 1 if *nIndloc* is greater than *nInd*. In such a case, the population locus ties together several individual loci from the same individual.

lon_over: longest possible overlap for any allele of the locus. 0 if the program did not try to merge the locus or if it could not merge.

lon_com_over: longest overlap possible for all alleles of the locus. The program used this overlap to merge. 0 if it did not try to merge or could not merge.

merge_conflict: 1 if lon_over > lon_com_over, 0 if not.

`popall.txt`

Format: tab-delimited text table with header line. One line per allele:

poplocID: ID of this locus

popall_ID: ID of this allele

popall_seq: the allele sequence. When the program merged overlapping fragments of this locus, the merged allele sequence.

popallvar: If the locus has only one allele: “consensus”, if it has several alleles, the characters of this allele at the variable positions, e.g. TGT in a locus with three SNPs.

popall_seq_notmerged: original allele sequence. Sequence before merging when the program merged overlapping fragments of this locus.

Algorithm details

Pairwise comparison of allele sequences and network building

The program uses algorithms similar to those in **indloc** to identify pairs of sequences that could have a distance <= *interind_d* to determine pairwise distances and to cluster sequences into networks. The algorithms are simpler because the input allele sequences determined by **indloc** do not contain N.