GlbPSs 1.0 Manual Part 11: Documentation of pair_finder_07.0.pl

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

This program identifies pairs of loci that have been identified based on paired sequencing reads from *rc-duplicates*. Two loci A and B are a pair when one read of a pair of reads has been assigned to locus A and the other to locus B. **pair_finder** produces outfiles that can be read by **data_selector**. You can use **data_selector** to select one locus of each pair of loci.

Requirements before you use pair_finder

You must execute **fdm**, **indloc**, **poploc** and **indpoploc**. Next, you should use **data_selector** to exclude loci with a length below 32, use **indel_checker** to identify indel variant loci, use **depth_analyzer** to identify repetitive elements, and use **data_selector** to exclude indel variant loci. Then use **pair_finder**. You must have paired reads. The fastq-headers must consist of five fields separated by underscores. The fifth field must be different in forward and reverse reads. The other fields must be identical. Activate option *ch_h* in **fdm** to convert Illumina sequence headers to this format. The paired reads should be reverse complements of each other. Activate the appropriate options for the *rc-locus-pairs-strategy* in **fdm** to produce them. (See the documentation of **fdm** for details.)

pair_finder produces outfiles in a directory `pairs` within the main database directory. It produces a warning and stops execution if this directory already exists. Remove or rename it before you execute **pair_finder** a second time.

Usage

To start the program, enter

```
pair_finder_07.0.pl
```
or
```
perl pair_finder_07.0.pl
```

pair_finder looks up your selection in directory `export` and analyzes your selected subset of data. When it has finished execution, use **data_selector** and activate `select loci: 1 Paired loci: select one of each pair` to select one locus from each pair of loci.

pair_finder produces a report in file `pairs/pairs_rep.txt`. When it analyzes a selected subset of data, it copies the selection report from file `export/sel_rep.txt` into this file. When you use **data_selector** to select one locus of each pair, it will notice that you have analyzed a selected subset of data with **pair_finder** and ask you to make sure that the respective selection of data is still valid (i.e. has not changed).

pair_finder reads in several files for each individual. One of these files can be a plain text file (`*_reads.txt`) or a gzip compressed file (`*_reads.txt.gz`), produced by **indloc**. **pair_finder** first searches for a plain text file, then for a gzip compressed file and reads in the first file it finds. Reading in gzip compressed files requires a bit more time than reading in plain text files.

Algorithm overview

pair_finder looks up the respective individual locus for every single read in the database or selected subset of data. If the individual locus is valid (if **indloc** identified at least one allele), **pair_finder** looks up the population locus it has been assigned to by **indpoploc**. The program identifies pairs of population loci that are connected by pairs of reads and counts the connecting read pairs and individuals. Both population loci must be in the selected subset of data to be identified as a pair.

Next, **pair_finder** clusters population loci into groups. Both loci of a pair are assigned to the same group. Most groups will contain two loci i.e. complete locus pairs. **pair_finder** can find groups with more than two loci when the paired reads are not *rc-duplicates* (not advisable). This should not happen, when you follow the recommended workflow.

A few groups may contain single loci. This happens because you already have excluded their partners as potential indel variants identified by **indel_checker** and USEARCH. It is conservative to assume that the remaining single loci could also be indel variants, which remained undetected. You will automatically exclude them in the next step when you use **data_selector** to select one locus of each pair. **data_selector** evaluates only groups containing two loci when you select one locus of each pair. It excludes all loci in groups of one or more than two loci.

<u>Split loci</u>

It can happen that **indpoploc** assigns different alleles of one individual locus to different population loci, i.e. splits up an individual locus. **pair_finder** excludes split loci from the analysis. They do not appear in any outfile out of **pair_finder**. **data_selector** automatically excludes them when you select one locus of each pair.

<u>Special cases: self-matching loci</u>

It is possible that a particular locus is fully or nearly palindromic. That means, the sequence of a given allele is so similar to its own reverse complement that **indloc** assigns the forward and the reverse complementary version of the same allele to a single locus. I call such loci self-matching loci here. It depends on the distance settings in **indloc** how similar a sequence must be to its own reverse complement to match it. It is thus also possible that one allele of a locus matches its reverse complement and another does not. With such variation, it is possible that an individual is homozygous for a self-matching allele A, another individual homozygous for a non-self-matching allele B and yet another individual heterozygous with alleles A and B. Depending on the distance settings, **indloc** will then identify one locus in the first individual and two loci in the other two individuals. Depending on the distance settings again, **poploc** will identify two corresponding population loci. Locus 1 will contain both versions of allele A and one version of allele B. Locus 2 will contain the other version of allele B.

You should not use self-matching loci for genotyping. The forward and the reverse complementary version of the same allele may appear as two different alleles in such loci.

pair_finder identifies self-matching loci in two steps: It identifies loci that contain any complete read pair as self-matching. Next, it identifies all loci that are connected to a self-matching locus through any read pair. That means, in the example above, it will treat locus 1 and locus 2 as self-matching loci. The program stores the IDs of self-matching loci separate from other loci in an outfile `pairs/self_match.txt`. **data_selector** automatically excludes self-matching loci when you select one locus of each pair.

Outfiles

pair_finder produces four outfiles, `pairs_rep.txt`, `pairs.txt`, `loc_groups.txt` and `self_match.txt` in a directory `pairs` within the main database directory. Do not modify or delete these files when you plan to use **data_selector** to select one locus of each pair.

`pairs_rep.txt`

This file contains a report with error messages (if any), and summary counts of locus groups with different numbers of loci and self-matching loci. The first line says if you analyzed the whole database or a selected subset of data.

`pairs.txt`

Format: table with header line in a tab-delimited text file. This file lists pairs of loci (poplocID1 and poplocID2) and the number of connecting read pairs (n_seqpairs) and individuals (n_ind).

`loc_groups.txt`

Format: table with header line in a tab-delimited text file. This file lists the identified groups of loci, which are consecutively numbered (loc_group), the number of loci in each group (n_loc) and the ID of each locus in a group (poplocID).

`self_match.txt`

Format: table with header line in a tab-delimited text file. This file lists self-matching loci and loci connected to self-matching loci. It contains the locus ID (poplocID) the number of self-matching read pairs (n_seqpairs) and the number of individuals (n_ind) these read pairs stem from. The values of the latter two variables can be zero when a locus contains no allele that matches its own reverse complement. Such loci are nevertheless treated as self-matching loci when they are connected to a self-matching locus by any read pair.

Infiles

The program needs the following infiles. It produces an error message and stops execution when it cannot open a file.

Infile	produced by
`export/sel_ind.txt`	data_selector
`export/sel_loc.txt`	data_selector
`export/sel_gt.txt`	data_selector
`export/sel_rep.txt`	data_selector
`individuals.txt`	indloc
`*_reads.txt / *_reads.txt.gz`	indloc
`poploc.txt`	poploc
`split_loci.txt`	indpoploc
`*_indpoploc.txt`	indpoploc

*_ stands for an individual ID