GIbPSs

Genotype **I**ndividuals **b**ased on **P**aired **S**equences *or* ***s****ingle reads*

Version 1.0.0

Manual Part 1: Package overview

5/22/2015

Andreas Hapke

Johannes Gutenberg University Mainz

Institute of Anthropology

Anselm-Franz-von-Benzel-Weg 7

D-55099 Mainz, Germany

ahapke2@gmail.com

Table of contents

Introduction

GIbPSs is a package for the analysis of GBS (genotyping-by-sequencing) and RAD (restriction site associated DNA) sequence data. Its main purpose is genotyping of individuals from diploid organisms without a reference genome for population genetic and phylogeographic analyses. GIbPSs is written in Perl 5 and runs under Linux and Windows. I have written it for the analysis of paired sequencing reads from a "classical" GBS library (Elshire et al. 2011). GIbPSs is not restricted to this datatype. It handles single or paired reads from protocols such as GBS, two-enzyme GBS, and ddRAD. It can also analyze forward reads from RAD libraries. GIbPSs can handle sequences of different lengths, identify and analyze short restriction fragments in adapter-containing sequences and deal with indel variation within loci. GIbPSs identifies loci based on configurable distance criteria, which require testing. You can rapidly perform multiple runs of GIbPSs because it employs fast algorithms and can run several processes in parallel during time-consuming steps (see "Runtimes and file sizes"). Several programs in the package can read and write gzip compressed files, which saves a lot of disk space. GIbPSs stores genotype data in a self-containing database that you can easily archive or move to a different storage location. It offers multiple options to explore, analyze, and filter your data, to select subsets of data and to export them in numerous formats.

This document gives you an overview over GIbPSs, installation instructions, a tutorial for a quick exploration of GIbPSs with simulated data, and instructions for getting started with real data. I focus on paired end sequence data from classical GBS libraries (Elshire et al. 2011) to explain the functionality of GIbPSs. See "Use of GIbPSs with other GBS protocols" for instructions how to use it with other data.

Paired end sequencing of classical GBS libraries

The properties of classical GBS libraries (Elshire *et al.* 2011) provide some challenges for data analyses without a reference genome. GIbPSs offers a special strategy for such data that relies on paired sequence reads. The technique of Elshire et al. (2011) uses a single restriction enzyme and two kinds of adapters, a set of barcode adapters and one common adapter (Fig. 1a). Adapters are ligated to restriction fragments in individual samples; samples are pooled and PCR-amplified with tailed primers. The PCR amplification enriches the library for short fragments. Almost all sequences generated from the library stem from fragments with two different adapters (barcode and common). These adapters bind at random at the ends of restriction fragments. Consequently, a locus (RAD tag) is sequenced on both strands based on different corresponding restriction fragments with a barcode adapter at one or the other end (Fig. 1b, A and B).

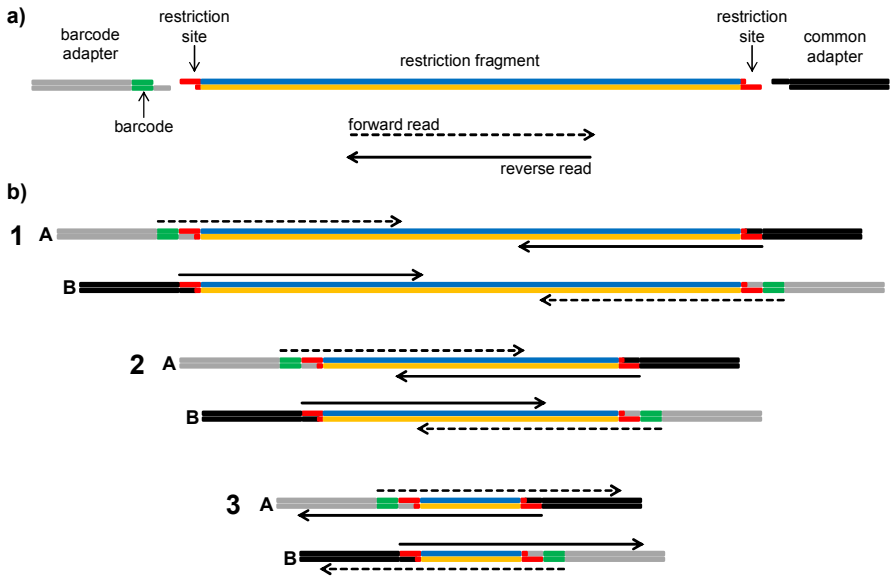

Fig. 1 GBS library: restriction fragments, adapters, sequencing reads

Reads from both strands of a locus do not overlap when the locus is longer than twice the read length minus the barcode length (Fig. 1b 1). They overlap partially when the locus length is between one and two read lengths minus the barcode length (Fig. 1b 2). Sequences overlap completely and contain adapter sequences when the locus is shorter than the read length (Fig 1b 3). With only forward reads and without a reference genome, it is almost impossible to find out, which sequences stem from the same locus when the sequences do not overlap.

Locus identification based on such data yields pairs of linked loci. Two loci in a pair are more or less overlapping fragments of two reverse complementary versions of the same locus. I call such pairs *rc-locus-pairs* here. Paired-end sequencing offers the possibility to identify *rc-locus-pairs* even when paired reads do not overlap. GIbPSs uses a fast and reliable strategy that I call *rc-duplicates-strategy* to identify *rc-locus-pairs.* It uses sequence information from both strands to identify a locus and determine the sequences of its alleles. That means, even when the read depth from each single strand is insufficient for genotyping, GIbPSs can use the sum of both and thus effectively exploit the full coverage in the data.

Package overview

GIbPSs includes 15 programs (Fig. 2), which I briefly present here. Each program has an own, detailed documentation. The recommended workflow in Figure 2 is optimal for paired read data from classical GBS libraries. See "Use of GIbPSs with other GBS protocols" for workflows with data from other GBS protocols.

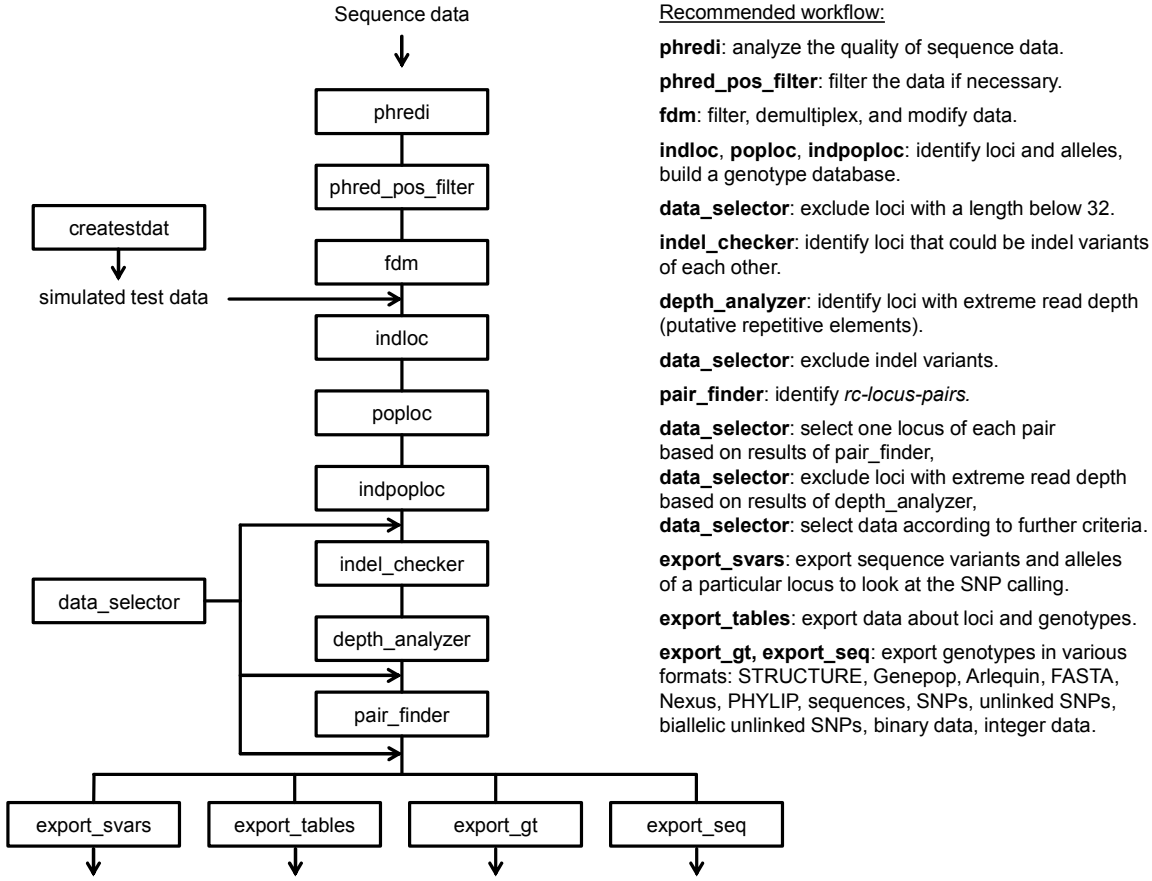

Fig. 2 Package overview and recommended workflow. The program createstdat creates a test dataset that you can use for a quick exploration of GIbPSs. See the tutorial in this document.

The first program, **phredi**, performs a quality check of your sequence data. If the data contain systematic error at specific positions, you can filter them with **phred_pos_filter**.

The next program, **fdm**, filters, demultiplexes and further modifies your sequence data. It can filter and sort sequences according to quality and other criteria, identify barcodes and restriction sites, repair them if necessary, demultiplex the data and truncate sequences. Together with the next program, **indloc**, **fdm** can use paired reads to construct pairs of reverse complementary sequences, which I call *rc-duplicates* here. These *rc-duplicates* are the basis for later identification of *rc-locus-pairs*. The construction of *rc-duplicates* requires different procedures for sequences that do not contain adapters (Fig. 3) and for sequences from short restriction fragments that contain adapters (Fig. 4).

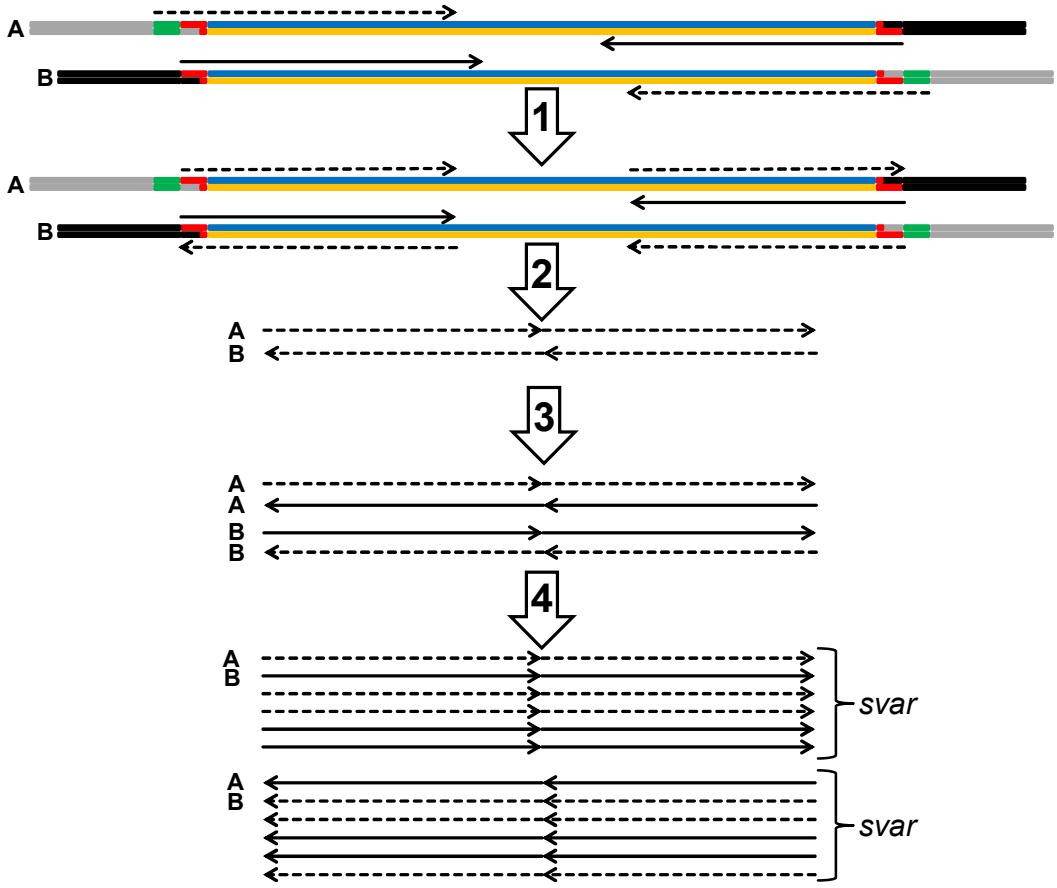

Fig. 3 Construction of *rc-duplicates* from sequences without adapter. Top: Fragments corresponding to a given locus carry the barcode adapter on one or the other end (A and B). The program **fdm** constructs *rc-duplicates*: Step 1: Identify and remove the barcode from the forward read, truncate both reads to equal length (configurable), determine the reverse complement of the reverse read. Step 2: Append the reverse complement of the reverse read to the forward read. Step 3: build a new reverse read as reverse complement of the new, elongated forward read. (Step 3 can also be performed by **indloc**, which saves disk space.)

Step 4. The program **indloc** identifies loci within individuals. It starts by collapsing identical reads from different fragments into sequence variants (*svar*) with a given depth. The forward read of fragment A is identical to the reverse read of fragment B and both are collapsed into the same *svar* together with other sequences.

The concatenated sequences in Fig. 3 consist of fragment that may in fact overlap. The program **poploc** later identifies overlapping fragments and merges them. This step happens after identification of all alleles of a locus in all individuals.

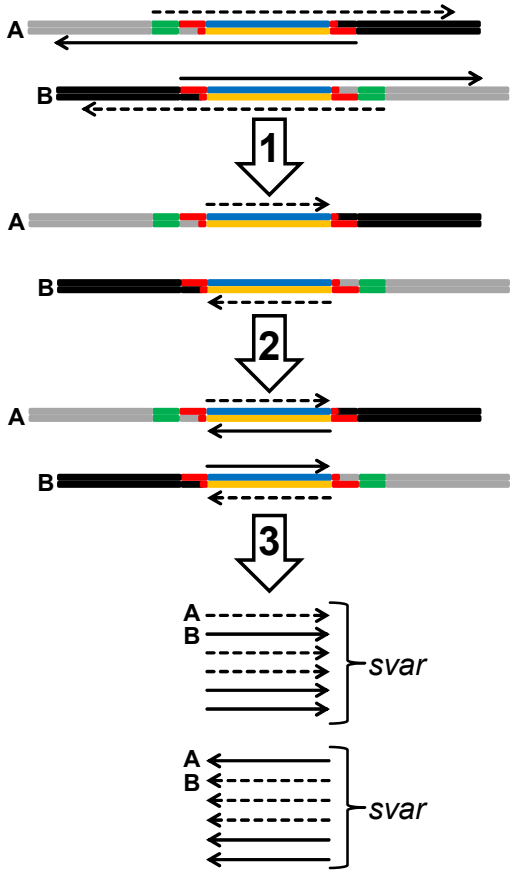

Fig. 4: Construction of *rc-duplicates from sequences with adapter* Top: Fragments corresponding to a given locus carry the barcode adapter on one or the other end (A and B). The program **fdm** has detected parts of the common adapter in forward reads and constructs *rc-duplicates*: Step 1: Identify and remove the barcode from the forward read, remove the adapter and all characters that are part of the restriction site at both ends. Step 2: build a new reverse read as reverse complement of the forward read. Step 2 can also be performed by the next program, **indloc**, to save disk space. **fdm** can also detect barcode adapters in reverse reads.

Step 3. The program **indloc** identifies loci within individuals. It starts by collapsing identical reads from different fragments into sequence variants (*svar*) with a given depth. The forward read of fragment A is identical to the reverse read of fragment B and both are collapsed into the same *svar* together with other sequences.

The identification of loci starts within individuals, i.e. in separate analyses of sequences from different individuals. The program **indloc** identifies identical sequences and collapses them into sequence variants (*svar*) with a given depth (Step 3, Fig. 3 and 4). Next, it clusters similar *svars* into networks based on configurable distance and read depth criteria (Fig. 5). Each network is the basis for an individual locus. The program stores all sequences of a locus in a matrix with sequences as rows and positions as columns. It analyzes each variable position and verifies if it represents sequencing or polymerase error or a SNP. Two SNP calling algorithms are available: a frequency threshold method and a binomial likelihood ratio method (Glaubitz *et al.* 2014). The program identifies and corrects errors in two steps - separately at each position and then by row wise analysis of the matrix. It determines alleles, which are phased haplotypes when a locus contains several SNPs. Each locus and each allele obtain an ID in each individual (indlocID, indall_ID).

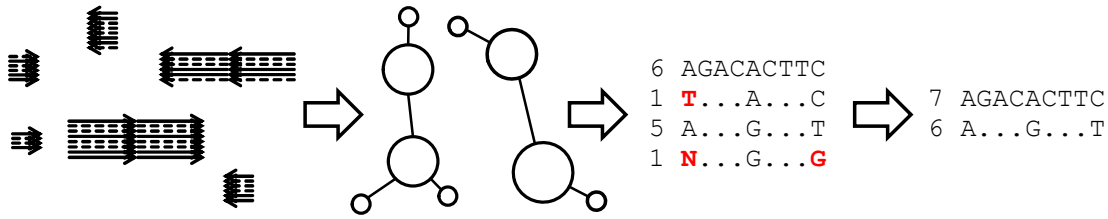

Fig. 5 Locus identification within individuals. The program **indloc** clusters similar *svars* into networks, corrects polymerase and sequencing errors and identifies allele sequences.

The next program, **poploc**, identifies loci based on the total sample ("population") of individuals in the dataset. It analyzes the sequences of all alleles from all individuals and clusters them into loci according to configurable distance criteria. After having identified all alleles of a locus in all individuals, **poploc** searches for overlapping fragments in allele sequences that stem from concatenated reads (Fig. 3). It analyzes all alleles of a locus and merges them to the same length. The reasoning behind this is that several overlaps may be possible between the two fragments of a given allele. Possible overlaps can differ between different alleles of a locus. Merging at an earlier stage would thus provoke the risk to assign alleles of the same locus to different pseudo loci. Next, **poploc** determines a consensus sequence, variable positions and other data for each locus. It assigns an ID to each locus (poplocID) and allele (popall_ID).

The program **indpoploc** links the results from **indloc** and **poploc**. It determines the corresponding locus (poplocID) and allele (popall_ID) for each locus (indlocID) and allele (indall_ID) in each individual. The combined outfiles from **indloc**, **poploc,** and **indpoploc** constitute a database with loci, alleles, genotypes, read depths, and many other data.

The next programs in the package use this database to perform additional analyses, select and export data. I present them here according to the recommended workflow in Figure 2.

data_selector

This interactive tool enables you to select subsets of data from the database. It offers various quality filters and criteria to select individuals, loci and genotypes. Use it now to exclude all loci with a length below 32 nucleotides if there are any in your database. This is a restriction of USEARCH (Edgar, R.C. 2010), which is used by the next program:

indel_checker

The programs **indloc** and **poploc** employ fast algorithms for the identification of loci. These algorithms ignore the problem that alleles of some loci may differ by indel variation. Consequently, such alleles are assigned to different loci. These loci are in fact pseudo loci, which are indel variants of each other. In a population genetic dataset, this problem will affect a relatively small fraction of loci. Nevertheless, it is desirable to identify the resulting pseudo loci and to remove them from the final dataset for downstream analyses. **indel_checker** identifies loci that could be indel variants of each other. It exports data from the database and calls the program USEARCH (Edgar, R.C. 2010) to analyze them. The output from **indel_checker** can be used by the next program, **depth_analyzer**, for an improved analysis of the sequencing depth across loci. Later, you will exclude the indel variant loci identified by **indel_checker** with the aid of **data_selector**.

depth_analyzer

A GBS library can contain repetitive elements or fragments of repetitive elements. Usually, **indloc** and **poploc** will assign sequences of different copies of a repetitive element to one single locus. Such pseudo loci are not useful for genotyping. Extreme sequencing depth is one clue to the identification of putative repetitive elements. This program analyzes sequencing depths of loci. You can use the results to define criteria that identify loci of extreme depth. You can later use **data_selector**, apply these criteria and exclude the respective loci.

data_selector: Use it now to exclude indel variant loci identified by **indel_checker** and USEARCH.

pair_finder

This program identifies *rc-locus-pairs*. These pairs are still unknown because **indloc**, **poploc**, and **indpoploc** treated the two loci of each pair as different loci.

data_selector: Use it now to select one locus of each *rc-locus-pair*.

data_selector: Use it now to exclude loci with extreme sequencing depth based on your analysis with **depth_analyzer**.

data_selector: Finally, you can apply various further criteria to select and filter individuals, loci and genotypes. The programs **export_tables**, **export_gt** and **export_seq** can then export your selected data.

export_svars: This program enables you to have a look at the SNP calling at a particular locus in a particular individual. It exports a FASTA file containing all occurring sequence variants (*svars*) of this locus and the inferred alleles.

export_tables

This program exports two tables with data about your currently selected subset of data. It produces two files with data about loci, such as a consensus sequence, the number of alleles, variable positions etc. One of these files describes properties of loci, as they are stored in the database. For example, the number of alleles is the number of all known alleles. The second file describes properties of loci within the currently selected subset of data. For example, the number of alleles includes only alleles included in the selection, and the consensus sequence is the consensus of these alleles. Two additional files contain all selected genotypes and the read depths of each allele.

export_gt

This program exports selected genotypes in several popular formats than can be read by population genetic and other programs: STRUCTURE, Genepop, Arlequin, FASTA, Nexus, and PHYLIP. It can export genotypes based on all known SNPs of a locus, export unlinked SNPs based on one SNP per locus, and restrict unlinked SNPs to biallelic unlinked SNPs. It produces FASTA, Nexus and PHYLIP files with DNA characters, binary data, and integer data. Several options enable you to modify the format of these files to facilitate the analysis with other programs.

export_seq

This program exports complete sequence data of each selected locus in FASTA, Nexus or PHYLIP format. It produces three files per locus: An alignment of individual sequences contains two sequences per individual. An alignment of nonidentical sequences contains all selected alleles of the locus. An additional file contains information about the number of copies of each allele in each individual. Several options enable you to modify the format of these files to facilitate the analysis with other programs.

Use of GIbPSs with other GBS protocols

Numerous modifications of GBS (Elshire et al. 2011) and RAD sequencing (Baird et al. 2008) exist. GIbPSs is very flexible and can be used for data from a broad variety of methods. There is one restriction: GIbPSs is designed for genotyping without a reference genome. It cannot use a reference genome. Here are some examples for use of GIbPSs with other GBS protocols. You may contact me by email when you need further advice.

Protocols with two different restriction enzymes and without random sharing (e.g. double digest RADseq, Peterson et al. 2012; two-enzyme GBS, Poland et al. 2012):

These protocols generate sequences from restriction fragments with two different cut sites on both ends. Forward reads from these fragments are all from the same strand of a locus. That means, you

do not need the *rc-duplicate-strategy* and you do not need reverse reads. If you have single reads from such libraries, follow the workflow for single reads below. If you have paired-end reads, follow the workflow for paired reads without *rc-duplicate-strategy* below. If some of your restriction fragments may be shorter than the read length and you wish to analyze them, configure **fdm** to crop adapter containing sequences and to save them into one outfile together with not-adapter-containing sequences.

Protocols with one restriction enzyme and random sharing (e.g. RAD, Baird et al. 2008)

GIbPSs can analyze single (forward) reads from such libraries. Usually, you will perform a size selection in your protocol. If it is possible that nevertheless some sequenced RAD tags are shorter than the read length, you must search for adapters in your sequences and discard the respective sequences. Follow the workflow for single reads below. Additionally, configure **fdm** to save adapter containing sequences to separate outfiles. Analyze only the sequences without adapter with **indloc**.

Genotyping of pooled DNA samples from several individuals

GIbPSs is made for genotyping of individuals from diploid organisms. Another approach is to pool DNA from several individuals from each population with one barcode, e.g. to search for fixed differences between populations. I could imagine that you can use GIbPSs with such data if you use the frequency threshold SNP calling method in **indloc**. See the documentation for details.

Workflow for single read data

Use these programs in this order. Please refer to the documentation of each program for details.

phredi: Analyze the quality of your sequence data.

phred_pos_filter: Filter them if necessary.

fdm: Filter, demultiplex, and modify your data. Configure fdm for analyses of single reads.

indloc: Identify loci in individuals. Do not build r-reads of *rc-duplicates*: use `-r 0` (default).

poploc: Identify loci across individuals. Do not merge sequences. Use `-msl 0` (default).

indpoploc: Link locus and allele IDs, build a genotype database.

data_selector: Exclude loci with a length below 32 if any.

indel_checker: Identify loci that could be indel variants of each other.

depth_analyzer: Identify loci with extreme read depth.

data_selector: Exclude indel variant loci based on results of indel_checker.

data_selector: Exclude loci with extreme read depth based on results of depth analyzer.

data_selector: Select data according to further criteria.

Use **export_svars**, **export_tables**, **export_gt**, and **export_seq** to export data.

Workflow for paired read data without *rc-duplicates-strategy*

Use these programs in this order. Please refer to the documentation of each program for details.

phredi: Analyze the quality of your sequence data.

phred_pos_filter: Filter them if necessary.

fdm: Filter, demultiplex, and modify your data. Configure fdm for analyses of paired reads, construct *rc-duplicates*, save only the f- reads of *rc-duplicates* (`save_f 1, save_r 0, r1r2sf 0`).

indloc: Identify loci in individuals. Do not build r-reads of *rc-duplicates*: use `-r 0`.

poploc, **indpoploc**: Identify loci and alleles, build a genotype database.

data_selector: Exclude loci with a length below 32 if any.

indel_checker: Identify loci that could be indel variants of each other.

depth_analyzer: Identify loci with extreme read depth.

data_selector: Exclude indel variant loci based on results of indel_checker.

data_selector: Exclude loci with extreme read depth based on results of depth analyzer.

data_selector: Select data according to further criteria

Use **export_svars**, **export_tables**, **export_gt**, and **export_seq** to export data.

Installation instructions

All programs of GIbPSs are written in Perl 5. They run under Windows and Linux. I have tested GIbPSs with Strawberry Perl 5.16.2 under Windows 7 and with Perl 5.18.2 under Ubuntu 14.04 LTS.

Download GIbPSs, unpack the archive, and store it anywhere on your computer. Directory `docu` contains the documentation. Directory `source` contains the programs. Each program is a single file that you can copy to a different location and execute there. Several programs need additional text files with settings that you must prepare for your analyses. Directories `user_files_Lin` and `user_files_Win` contain example files for Linux and Windows. Under Linux, all text files read by GIbPSs must have Linux line endings. If you use Strawberry Perl under Windows, they may have Linux or Windows line endings. The program files themselves have Windows line endings but they run under Windows and Linux.

Perl must be installed on your computer. If you use Linux, it is probably already installed. If you use Windows, you can download Strawberry Perl at http://strawberryperl.com/.

To check, which version of Perl is installed, enter the following in a command prompt:

```
perl -v i
```

GIbPSs needs the following Perl modules:

IO::File
IO::Uncompress::Gunzip
IO::Compress::Gzip
List::Util
Math::Round
Parallel::ForkManager

Some of them may not yet be installed on your system. To check if a module is installed, enter the following (here for Parallel::ForkManager) in a command prompt:

```
perl -MParallel::ForkManager -e 1
```

If there is absolutely no output, the module is installed. If you get an error message, it is not.

To install a module from CPAN, enter the following in a command prompt:

```
cpan Parallel::ForkManager
```

and follow the instructions on screen. If you use cpan for the first time, the instructions will help you to configure cpan. Possibly, you do not have the necessary privileges to install a module. One of several options under Linux is to configure cpan to use sudo during the install phase. For more information about installing modules from CPAN under Linux and Windows, see: http://www.cpan.org/modules/INSTALL.html.

USEARCH (Edgar, R.C. 2010) must be installed on your system. The program **indel_checker**, which is part of GIbPSs, uses it to identify loci with indel variation. USEARCH is available for Linux, Windows and Mac OSX at http://www.drive5.com/usearch/. I have tested **indel_checker** with USEARCH v7.0.1090.

License information

GIbPSs is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

GIbPSs is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License version 3 in a file License.txt along with GIbPSs. If not, see http://www.gnu.org/licenses/.

How to cite GIbPSs

A manuscript about GIbPSs is in preparation. For the time being, please cite GIbPSs as follows:

Hapke, A. (2015) GIbPSs: GIbPSs version 1.0.0. Available from https://github.com/ahapke/gibpss.

The version number of GIbPSs will change in future releases. Please adopt it accordingly.

Tutorial: Quick Exploration of GIbPSs with simulated data

This tutorial enables you to perform a quick test of the software. Please refer to the documentation of each program for details. The names of the programs of GIbPSs contain version numbers, which can change in future releases. Please adopt them accordingly when you call a program as described below.

Create a new directory somewhere on your computer, e.g. `testdatabase`. At the end of the tutorial, this directory will have a size of approximately 500 Mb.

Copy all program files of GIbPSs from directory `source` into the new directory. (You may leave out **phredi**, **phred_pos_filter**, **fdm** and **depth_analyzer** - you will not use them now.)

Open a command prompt and cd to the directory.

Call the first program, **createstdat**, as follows:

```
createstdat_03.0.pl
```

If that does not work, call **createstdat** (and other programs of GIbPSs) as follows:

```
perl createstdat_03.0.pl
```

This program creates a simulated test dataset and produces fastq files. The fastq files correspond to those that you would obtain with **fdm**, when you analyze paired sequencing data from a classical GBS library with the *rc-duplicates-strategy*. The simulated data have much less complex properties than real data but enable you to test many functions of the software quickly.

The simulated dataset comprises 1200 loci and 20 individuals. All loci have a length of 120 nucleotides and 2 to 6 different alleles, which differ by up to 6 SNPs. 200 loci contain indel variation: one allele has a deletion of 3 nucleotides at a randomly selected position. **createstdat** simulates individual genotypes and samples 40 read pairs per individual and locus. It samples reads from the two alleles of a locus at random. Forward- and reverse reads stem from one or the other DNA strand

with a probability of 0.5. All reads have a length of 80 nucleotides and contain errors with a frequency of 0.001. The program constructs *rc-duplicates* and saves their forward reads to fastq-files.

The program creates a subdirectory `testdata` that contains the fastq files.

It creates a subdirectory `user_files` that contains files with suitable distance settings for **indloc** and **poploc** and a file for **indloc** about the fastq files.

Each of the following programs will create files within directory `testdatabase`, some of them in new subdirectories.

Call **indloc** as follows. Use a smaller value for flag -P when you wish to perform less than 4 processes in parallel.

```
indloc_22.0.pl -i user_files/inds_infiles.txt -d user_files/indloc_dist.txt
-D 4 -P 4 -r 1 -M f
```

Call **poploc** as follows:

```
poploc_09.0.pl -d user_files/poploc_dist.txt -msl 160 -mino 10 -spl 117
```

Call **indpoploc:**

```
indpoploc_06.0.pl
```

Call **indel_checker**:

Use appropriate values for your system for flags -UP and -DP. -UP specifies the full path to USEARCH, -DP the full path to your database directory. Example:

```
indel_checker_03.0.pl -id 0.85 -UP C:\myprograms\usearch\usearch.exe -DP
C:\data\testdatabase
```

Normally, you should now run **depth_analyzer**. We skip this step because all loci have the same sequencing depth.

Call **data_selector** to exclude loci that could be indel-variants of other loci:

```
data_selector_16.0.pl
```

Select "`2 select loci`", then "`2 List of loci in a file`", provide this filename: "`indelcheck/no_indel_loc.txt`", select "`13 Run`".

Call **pair_finder**:

```
pair_finder_07.0.pl
```

Call **data_selector** to select one locus of each pair:

```
data_selector_16.0.pl
```

Select "`2 select loci`", then "`1 Paired loci: select one of each pair`", enter "`N`" to answer the question with no, select "`13 Run`".

It can be helpful to make a copy of directory `export` with a different name now. This directory has been created by **data_selector** and contains files about your current data selection. You can rename your copy to `export` again to restart a selection from the point where you are now.

Open file `sel_rep.txt` in directory export to have a look at your current selection of data. You will notice that you have selected approximately 1000 loci. That means, you have successfully identified and excluded all or most indel-variant loci, identified all *rc-locus-pairs*, and selected one locus of each

pair. Close the file again before you continue. It is possible that **indel_checker** and USEARCH could not find all indel-variant loci: Some indel-variation could have been lost due to allelic dropout during the simulation of sequence-reads.

Now you can try further selection criteria in **data_selector** and export data with **export_svars**, **export_tables**, **export_gt**, and **export_seq**. Please refer to the documentations of these programs for details.

Please select a small number of loci with **data_selector** before you use **export_seq** if you do not wish to get several thousand outfiles.

Getting started with real data

You need sequence data from a GBS library in a set of fastq-files. The program **fdm** can deal with fastq files in Illumina format. The fastq files may be plain text files or gzip compressed files. You need paired reads to use the *rc-duplicates-strategy*. Alternatively, you can analyze single reads. The sequences may contain undetermined bases (N) and bases with low Phred scores. They should not contain systematic errors, i.e. a certain fraction of sequences with N or low Phred scores at single specific positions.

Here, I give some short explanations how to organize the various files when you start working. Please read the documentation of each program before you start.

Store your sequence data anywhere on your computer.

Use **phredi** to check their quality. If necessary, use **phred_pos_filter** to filter your data. Use **phredi** again to verify the quality of the filtered data. Then filter, demultiplex and modify your sequence data with **fdm**.

For each of these steps, you should create a separate directory and store a copy of the program and additional files needed by the program therein. Then open a command prompt, cd to the directory and call the program as described in the documentation. It is not necessary to copy the sequence data around. Each program reads them where they are. You can instruct **phred_pos_filter** where to save the filtered fastq files. **fdm** will save demultiplexed fastq outfiles in a subdirectory. Each program can read sequence data as plain text files or gzip compressed files. **phred_pos_filter** and **fdm** can produce fastq files as plain text files or gzip compressed files.

Now, create a new directory for your first database, e.g. `main_database`. Place copies of **indloc**, **poploc**, **indpoploc**, **data_selector**, **indel_checker**, **depth_analyzer**, **pair_finder**, **export_svars**, **export_tables**, **export_gt**, and **export_seq** in `main_database`. All programs will produce outfiles within `main_database`, some will create subdirectories and store outfiles therein. Outfiles from one program are needed as infiles for another and must remain where they have been stored. It is however no problem to move the whole directory `main_database` to a different location.

Some of these programs will need additional text files with user settings. See the documentations of the programs. You can store these files within `main_database` or create a subdirectory (e.g. `main_database/user_settings`) and store them therein.

Follow the recommended workflow and start with **indloc**. Indloc will analyze one fastq file per individual. These files are outfiles out of **fdm**. You can instruct **indloc** with an additional file where to find them. From here on, all infiles and outfiles will be in directory `main_database`.

Runtimes and file sizes

Runtimes depend on many factors: size, quality, and properties of your sequence data, your settings in GIbPSs, and your hardware. To give you some orientation, I present some data from example analyses here (Table 1). I analyzed an empirical dataset with paired sequences from 12 individuals from a classical GBS library. The dataset comprised 65.7 Mio sequences with a length of 100 nucleotides. I used a desktop computer with an Intel(R) Core(TM) i5-3470 CPU @ 3.20 GHz x 4 with 16 GB RAM, and I used four cores with **fdm** and **indloc**. I demultiplexed, filtered and modified the data with **fdm**. Then I analyzed them according to the recommended workflow from **indloc** up to **pair_finder**. The export of data with the remaining programs requires just a few seconds of runtime. The programs **phredi**, **phred_pos_filter**, **fdm**, and **indloc** can read and write gzip compressed files. This saves a lot of disk space but makes computations a bit slower. I performed two series of analyses (Table 1): A: gzip compressed files: The input data were gzip compressed and **fdm** and **indloc** wrote gzip compressed files. B: text files: The input data were plain text files and **fdm** and **indloc** wrote plain text files.

Table 1: Runtimes and file sizes: analysis of 65.7 Mio sequences from 12 individuals

	A: gzip compressed files	**B: text files**
fdm runtime	69 min	52 min
indloc to pair_finder runtime	61 min	42 min
Total runtime	**130 min**	**94 min**
Input sequence data file size	6.37 GB	23.5 GB
fdm output file size	3.33 GB	7.41 GB
File size database	1.22 GB	4.5 GB
Total file size	**10.92 GB**	**35.41 GB**

References

Baird, N.A., Etter, P.D., Atwood, T.S., Currey, M.C., Shiver, A.L., Lewis, Z.A., Selker, E.U., Cresko, W.A., Johnson, E.A. (2008) Rapid SNP discovery and genetic mapping using sequenced RAD markers. *PLoS ONE* 3 (10): e3376.

Edgar, R.C.(2010) Search and clustering orders of magnitudes faster than BLAST, *Bioinformatics* 26(19), 2460-2461.

Elshire, R.J., Glaubitz, J.C., Sun, Q., Poland, J.A., Kawamoto, K., Buckler, E.S., Mitchell, S.E. (2011) A robust, simple Genotyping-by-Sequencing (GBS) approach for high diversity species. *PLoS ONE* 6(5): e19379.

Glaubitz, J.C., Casstevens, T.M., Lu, F., Harriman, J., Elshire, R.J., Sun, Q., Buckler, E.S. (2014) TASSEL-GBS: A high capacity genotyping by sequencing analysis pipeline. *PLoS ONE* 9(2): e90346.

Peterson, B.K., Weber, J.N., Kay, E.H., Fisher, H.S., Hoekstra, H.E. (2012) Double digest RADseq: an inexpensive method for *de novo* SNP discovery and genotyping in model and non-model species. *PLoS ONE* 7(5): e37135.

Poland, J.A., Brown, P.J., Sorrells, M.E., Jannink, J.-L. (2012) Development of high-density genetic maps for Barley and Wheat using a novel two-enzyme genotyping-by-sequencing approach. *PLoS ONE* 7(2): e32253.