proovread manual

Thomas Hackl, Frank Förster

Contents

1 Build status

Last build	`https://api.travis-ci.org/BioInf-Wuerzburg/proovread.svg`
Master branch	`https://api.travis-ci.org/BioInf-Wuerzburg/proovread.svg?branch=master`
Develop branch	`https://api.travis-ci.org/BioInf-Wuerzburg/proovread.svg?branch=develo`

The build tests proovread under different Perl versions (5.10.1, 5.12, 5.14, 5.18, 5.20, 5.22) and samtools (1.2, 1.3).

2 What's new

1. proovread-2.14.0
    - Makefile install interface
2. proovread-2.13.13
    - Usage of SeqChunker v0.4, therefore support of MacOSX
3. proovread-2.0
    - much faster and more sensitive due to **bwa mem** as default mapper
    - increased speed and contiguity by using **unitigs** in correction
    - optimized for HiSeq & **MiSeq** reads
    - compressed BAM intermediates
    - efficient threading up to 20 cores and more
    - ...

3 Installation

```
# fresh install
git clone https://github.com/BioInf-Wuerzburg/proovread
(cd proovread && make && make install)
# make install takes: PREFIX=/my/software/proovread

# make a test run
(cd proovread && make sample)

# update existing install to latest version
(cd proovread && make update)
```

NOTE: proovread comes with its own, modified version of `bwa`. Using it with a standard bwa built will fail.

4 Dependencies

- Perl 5.10.1 or later
  - `Log::Log4perl`
  - `File::Which` (see #17)
- NCBI Blast-2.2.24+ or later
- samtools-1.1 or later

proovread is distributed ready with binaries of SHRiMP2 and BLASR. If you want to employ your own installed version of these mappers, have a look at the `Advanced Configuration` section.

5 Usage

Test your installation by running proovread on the included sample data set.

```
(cd proovread && make sample)
```

Split long read data into chunks. See `Getting chunk sizes right` for details, recommendations have changed a bit for recent versions.

```
# small genome (10Mbp) or small memory machine or short runtime limit
SeqChunker -s 20M -o pb-%03d.fq pb-subreads.fq

# large genome (>500Mbp) and >8 GB RAM, no runtime limit
# either go for one instance of proovread per SMRT cell or
SeqChunker -s 1G -o pb-%03d.fq pb-subreads.fq
```

Run proovread on one chunk first. See `Log and statistics` for some notes on how to interpret logging information.

```
proovread -l pb-001.fq -s reads.fq [-u unitigs.fa] --pre pb-001
```

If things went smoothly, submit the rest.

6 Input

6.1 long-reads

Primarily proovread has been designed to correct *PacBio subreads* (see for details on PacBio terminology). You get these reads either from PacBio's SMRT-Portal or by using `dextract` from Gene Myers PacBio assembler DAZZLER, which I would recommend.

In general, reads can be provided in FASTQ or FASTA format. Quality information is used, but only has minor advantages. More valuable are subread information given in default PacBio IDs, which if available are utilized by proovreads `ccseq` module to improve correction performance. Reads shorter then 2x the mean short read length will be ignored.

It is also possible to feed other types of erroneous sequences to proovread, e.g. contigs, 454 reads, ... However, keep in mind that the alignment model for mappings has been optimized for PacBio reads and may produce artifacts in other scenarios. We are currently working on a version optimized for *Oxford Nanopore* data.

6.2 short-reads

For correction of long reads, proovread needs high coverage short read data. Typically these are HiSeq (75-150bp) and MiSeq reads (200-300bp), with overlapping libraries merged (FLASh) for best performance. But also 454 or PacBio CCS reads can be used.

Reads need to have FASTQ/A format and may differ in length. Pairing information are not used. Use of quality trimmed or error corrected reads can improve results.

The recommended coverage for short reads data is around 30-50X and should be specified with `-coverage`. If you have less coverage, it is definitely still worth running proovread. However, it is likely that contiguity will suffer.

Internally, proovread will sample subsets for different iterations, by default 15X for initial runs, 30X for the finishing. For customization of these rates see `sr-coverage` in proovread's config (`Advanced Configuration`).

6.3 unitigs

In addition to short reads, unitigs can be used for correction in particular for large data sets (eukaryotes). Unitigs are high-confidence assembly fragments produced by for example ALLPATHS, Meraculous2 or the Celera Assembler. In contrast to contigs, unitigs don't extend past any conflict in the underlying short read data, making them highly reliable.

Unitigs need to be used in combination with short read data for proovread to work properly.

Unitigs can improve contiguity - they usually are longer than short reads and, thus, easier to

align also in difficult regions - and decrease runtime - unitig computation removes most of the redundancy of the reads set. However, the effects of unitigs on accuracy and performance of proovread strongly varies between data sets. In particular on more complex data sets, i.e. large eukaryotic genomes with repeats, heterozygosity etc. use of unitigs can have a negative effect and reduce correction performance / quality. Unitigs, therefore, should be **used with caution** and the obtained results should be compared to runs without unitigs.

1. dazzling proovread - dazz2sam Currently, support for DAZZLER/DALIGNER is considered experimental. To use dazzler instead of blasr, either export paths or set `daligner-path` and `dazz-db-path` in the config and invoke with modes `sr+dazz-utg / mr+dazz-utg`. In the current implementation, only a single instance of dazzler will be invoked, therefore threading is determined by the thread setup with which daligner has been compiled (default 4).

   Since proovread is designed to operate on BAM/SAM, for the time being, daligner output is internally converted to SAM using a simple parser script (`dazz2sam`). This script also works as a stand-alone tool for dazzler-to-SAM conversion (`proovread/bin/dazz2sam -help`), which might come in handy if one wants to visualize dazzler mappings in common alignment viewers like IGV or tablet.

2. extracting unitigs from ALLPATHS

   ```
   # extract unitigs from allpaths assembly
   allpathslg/bin/Fastb2Fasta IN=reads.unibases.k96 OUT=unitigs.fa
   ```

7 Output

By default, proovread generates six files in the output folder:

.trimmed.f[aq]	high accuracy pacbio reads, trimmed for uncorrected/low quality regions
.untrimmed.fq	complete corrected pacbio reads including un-/ poorly corrected regions
.ignored.tsv	ids of reads and the reason for excluding them from correction
.chim.tsv	annotations of potential chimeric joints clipped during trimming
.parameter.log	the parameter set used for this run

If you are interested in mappings (BAM) and other intermediary files from iterations have a look at `-keep-temporary`.

The phred scores produced by proovread derive from short read support of each base during correction. The values are scaled to realistically mimic sequencing phred accuracies:

Phred	Accuracy	p33
40	99.99	I
30	99.90	?
20	99.00	5
10	90.00	+

8 Log and statistics

proovread generates a comprehensive log on STDERR. The includes fully functional system calls for scripts/tools invoked by proovread. That way, if something goes wrong, its easy to rerun a certain task individually and take a closer look on the issue.

If you want to analyze, how things are going and whether there might be problems with sensitivity etc., the most important information is `Masked: xx%` after each iteration.

```
grep -P 'Running mode|ked :|ning task' proovread.log
[Mon Jan 26 09:52:05 2015] Running mode: blasr-utg
[Mon Jan 26 09:52:51 2015] Running task blasr-utg
[Mon Jan 26 10:00:32 2015] Masked : 55.3%
[Mon Jan 26 10:00:32 2015] Running task bwa-mr-1
[Mon Jan 26 10:21:45 2015] Masked : 76.2%
[Mon Jan 26 10:28:14 2015] Running task bwa-mr-2
[Mon Jan 26 10:37:55 2015] Masked : 92.2%
[Mon Jan 26 10:39:46 2015] Running task bwa-mr-finish
[Mon Jan 26 10:51:19 2015] Masked : 93.0%
```

Masked regions are regions that have already been corrected at high confidence, minus some edge fraction, which remains unmasked in order to serve as seeds for subsequent iterations. After the first iteration, you should have a masking percentage $>$ 50-75%, strongly depending on quality, type and coverage of your data. With each iteration, this value should increase.

Prior to the final iteration, all data is unmasked and the final iteration is run with strict settings on entirely unmasked data. The obtained percentage can be slightly lower as in the last iteration, and is roughly equal to the amount of read bases that will make it to high-confidence .trimmed.fq output.

9 Advanced Configuration

proovread comes with a comprehensive configuration, which allows tuning down to the algorithms core parameters. A custom configuration template can be generated with `-create-cfg`. Instructions on format etc. can be found inside the template file.

10 Algorithm and Implementation

Algorithm and Implementation are described in detail in the proovread publication. An overview is given in on the proovread mechanism poster.

10.1 bwa-proovread

proovread does local score comparison, rather than using a single hard cut-off. bwa-proovread is modified in the same fashion. `proovread.[ch]` extend bwa with an implementation of proovread's binning algorithm. Reporting of alignments is determined by score-comparison within bins. That way repeat alignments are filtered early on, increasing performance and largely reducing disk space requirements.

11 FAQs and general Remarks

11.1 Getting chunk sizes right (#48)

Splitting your long read data into chunks of appropriate size is an important step to match your particular data set with proovread's correction procedure as well as with the computational hardware and infrastructure you are using.

Splitting can be carried out with proovread's `SeqChunker` program. It accepts chunk sizes in bytes, megabytes, gigabytes, ... (see `SeqChunker -help` for more options). One byte equals one character. Fastq files use one character per base (bp) plus on character for the quality of the base, and a few more characters are required for the header of each read. Hence, for fastq 2 bytes roughly transfer to 1 bp.

Rule of thumb for creating and running chunks: **Max size and threads for best performance, but with respect to the following limits:**

- Genome size: chunks should not contain more than 1X coverage of the genome, i.e. 10 Mbp genome: 20M chunk, 2GB genome: 4G chunks, ...
- RAM of machines: chunk size directly correlates with memory consumption. Try 8GB: <400M, 16GB: <800, ...
- Runtime limit: chunk size directly correlates with runtime. If you submit jobs to a scheduling system with per job runtime limits, decrease chunk size to match those limits if necessary.

Note, it always make sense, especially for large genomes/data sets to initially generate at least one small chunk and run proovread on this chunk to get an idea of overall performance.

```
# generate a single <10Mbp chunk
```

```
SeqChunker -s 20M -l 1 -o pb-%03d.fq pb-subreads.fq
```

11.2 What to expect from correction / Crunching numbers

Just to give you a hands-on idea of what to expect from correction. Here are some stats of the latest correction I ran. It's from one PacBio cell of a 50Mb heterozygous eukaryote genome (I will add some more numbers on other data sets and correction tools soon)

	raw	proovread	lordec
Sequences	56,877	55,493	53,676
Total (bp)	315,511,633	236,687,413	99,379,617
Longest (bp)	27,874	24,682	13,917
Shortest (bp)	1,000	1,000	1,000
N50 (bp)	7,592	6,236	1,877
N90 (bp)	2,887	1,934	1,141

11.3 Why do proovread results from two identical runs differ / Is proovread deterministic?

One might expect that proovread results are deterministic - meaning reproducible in identical form if input data is identical. This, however, is not the case in a couple of steps:

1. bwa mem mappings bwa employs heuristics that allow for slightly different results in repeated runs. In particular, one feature is prone to generate differences when employed in proovread's iterative strategy: for performance reasons bwa encodes nucleotides using 2 bits only, meaning bwa only has a four letter alphabet `[ATGC]`. Other bases, including `NNNN` stretches used for masking by proovread, are converted into random `[ATGC]` strings. This, in particular, effects alignments at the margins of masked regions:

```
   orig | ATGAATTGGTTAATCTGC
 masked | ATGAATTGGTNNNNNNNN
   read |    AATTGGTTAAT
        |
rand-01 | ATGAATTGGTAGCCATGG
        |    |||||||
 aln-01 |    AATTGGT
        |
rand-02 | ATGAATTGGTTTATCTGC
        |    |||||||| ||
 aln-02 |    AATTGGTTAAT
```

2. sorting with threshold Whenever there are decisions to make for sorted list in combination with fixed amount of items to keep/remove, things get non-deterministic if

identical values in sorting fields occur. In proovread, this for example affects filtering of "best alignments" in bins (localized scoring context).

3. consensus calling 50-50 ratios in base calling will result in one randomly chosen alternative, minimizing a particular bias.

11.4 Read IDs don't look like proper PacBio subread IDs / no-ccs mode (#27, #76)

proovread by default expects to work with PacBio subread data (see PacBio terminology / PacBio SMRT data). The reads have a standardized header as shown below, that provide information about the relative location of an individual subread within the full polymerase read. proovread scans header based on this convention to parse the subread coordinates, which are required to generate circular consensus sequences.

A *Read IDs don't look like proper PacBio subread IDs* warning will be thrown in case the provided data does not match this convention. **If your data are not PacBio subreads, you can ignore this warning**. proovread will simply skip the circular consensus step. If your data are subreads, then something is off with their format.

```
 m140415_143853_42175_c100635972550000001823121909121417_s1_p0/553/3100_11230
12 3 4567 8
```

1. `m` = movie
2. Time of Run Start (`yymmdd_hhmmss`)
3. Instrument Serial Number
4. SMRT Cell Barcode
5. Set Number (a.k.a. "Look Number". Deprecated field, used in earlier version of RS)
6. Part Number (usually `p0`, `X0` when using expired reagents)
7. ZMW hole number, unique read ID within cell
8. Subread Region (`start_stop` using polymerase read coordinates)

11.5 Chimeras, siamaeras and so on

1. Chimeras in general OK, let's try to clarify terminology. PacBio generates two types of chimeric reads:

    a) 1. unsplit subreads, i.e. siamaera Reads structure looks like this: `--R1-->-A-<-R1.rc-`. This happens quite frequently, however, it strongly depends on chemistry and particularly the quality of the libary prep. The more short DNA fragments make it

through size selection, the more chances for having multiple subreads in one read... I've seen libraries form <1% to >5%.

b) 2. random fusion chimeras I'm not exactly sure, when or how this happens, but there is a fraction of reads, where random sequences seem to be fused together. Probably some blunt end ligations during library prep, or similar effect ... This seems to happen quite rarely, and it is hard to quantify exactly, as there are other effects, that can cause reads to look like chimeras, although they aren't. For proper quantification, you would need a perfect reference (no collapsed copies, etc), your sample should not contain any structural variations, all reads would have to map uniquely, so no large genomes with repeats... And also correction itself can generate chimeras in-silico, e.g. if there are regions with multiple, yet slightly different copies in the sequenced genome, and this variation is lost in consensus steps.

2. chimera detection (short read based) There are three cases one needs to consider when looking at proovread's chimera detection based in short reads mappings:

a) 1. no alignments across breakpoint

```
:---->     <-----: short reads
:----->   <------:
:-------|--------: chimeric pacbio sequence
```

This happens often with siameras, because at the breakpoint, there is a (corrupted) adapter sequence that is not present in short read data. But also if original sequences and the sequence that was put their through fusion look rather different. In this case, it comes down to a) the sensitivity and scoring of the alignments and b) the mode of mapping (shrimp aligns global, bwa aligns local with a penalty on trimmed alignments, making breakpoint overlap alignments less likely.)

In any case, it is not possible to detect the fusion event. However, these regions will be removed during trimming, since there is no coverage for the consensus, hence, at least trimmed reads won't be chimeric.

b) 2. not enough inconsistencies in aligned reads

```
:-->  <A-G-------: short reads
:------A-G>   <--:
:------A|G-------: chimeric pacbio sequence
```

If original and fusion sequence look similar, you will get short read coverage at the breakpoint, but unless their are inconsistencies in the bases of aligned reads, the chimera will go undetected.

c) 3. detectable chimera

```
:-->  <C-G-------: short reads
:------A-T>   <--:
:------A|G-------: chimeric pacbio sequence
```

If there is short read coverage at the breakpoint and there are differences in reads overlapping from either side, proovread will call the chimera, report it to *.chim.tsv and split the reads accordingly when writing trimmed output (`SeqFilter -substr *.chim.tsv`). Details of detection and assessment of "differences" are described in more detail in the paper. But basically, I compare whether splitting would decrease overall error rates at the location or not.

3. siamaera detection (by long read structure) The siamaera module works independently of short read alignments and specifically targets improperly split subreads. This module wasn't part of the original publication, I added it, because in some of my data sets, quite a lot of this type of reads made it through correction, giving me trouble during assembly. It's neither pretty nor fast, but gets the job done. It works pretty reliable, however, there is a chance that palindromic regions get trimmed as well.

   When I introduced this filter, I also decreased sensitivity of proovreads internal "detect-chimera" filter. Original settings were rather hypersensitive - better-safe-than-sorry - working well for the paper data, but producing quite a lot of false positives on complex and particularly heterozygous data sets.

   Siamaeric reads usually are separated by short joint sequences (corrupted adapter). Detection is based on blasting and identifying reads with reverse complement self hits. Reads are trimmed to the longest non-chimaeric subsequence without joint sequence.

   ```
   ----R--->--J--<--R.rc--  siamaeric read
   ----R--->                trimmed read

   --R->-J-<----R.rc-----   siameric read2
           <----R.rc-----   trimmed read2
   ```

4. Running chimera and siamaera trimming stand-alone (#75) After the iterative correction, proovread runs trimming of low-quality regions, potential chimera and siamaera breakpoints all in one step (`Quality trimming and siamaera filtering raw output` in the log). To run chimera/siamaera indepedently quality trimming, you can use the following commands:

   ```
   # setup environment
   export PATH=/path/to/proovread/bin:$PATH
   export PERL5LIB=/path/to/proovread/lib:$PERL5LIB

   # proovread run data
   PRE=/path/to/run-prefix/prefix
   ```

```
# chimera trim only, no low-quality
SeqFilter \
 --in $PRE.untrimmed.fq \
 --phred-offset 33 \
 --min-length 500 `# proovread default` \
 --substr $PRE.chim.tsv \
 --out $PRE.chimera-only.fq

# siamaera trim only
siamaera \
 < $PRE.untrimmed.fq \
 > $PRE.siamaera.fq
```

12 Citing proovread

If you use proovread, please cite:

proovread: large-scale high accuracy PacBio correction through iterative short read consensus. Hackl, T.; Hedrich, R.; Schultz, J.; Foerster, F. (2014).

Please, also recognize the authors of software packages, employed by proovread:

Exploring single-sample SNP and INDEL calling with whole-genome de novo assembly. Li H. (2012) (bwa)

Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): application and theory. Mark J Chaisson; Glenn Tesler. (2012)

SHRiMP: Accurate Mapping of Short Color-space Reads. Stephen M Rumble; Phil Lacroute; Adrian V. Dalca; Marc Fiume; Arend Sidow; Michael Brudno. (2009)

13 Contact

If you have any questions, encounter problems or potential bugs, don't hesitate to contact us. Either report issues on github or write an email to:

- Thomas Hackl - thackl@lim4.de
- Frank Foerster - frank.foerster@uni-wuerzburg.de