seqOutBias v1.1.0 User Guide
2017/05/31

Contents

Introduction

Molecular biology enzymes have nucleic acid preferences for their substrates; the preference of an enzyme is typically dictated by the sequence at or near the active site of the enzyme. This bias may result in spurious read count patterns when used to interpret high-resolution molecular genomics data. The *seqOutBias* (https://github.com/guertinlab/seqOutBias) program aims to correct this issue by scaling the aligned read counts by the ratio of genome-wide observed read counts to the expected sequence based counts for each k-mer. The sequence based k-mer counts take into account mappability at a given read length using Genome Tools' Tallymer program. The *seqOutBias* program allows for flexibility in specifying the k-mer, including varying the k-mer size, k-mer information spacing, and specifying strand-specific offsets for the start of the sequence reads. Due to the large size of some datasets, *seqOutBias* reads compressed files (FASTA, mappability information, and BAM files), and reuses intermediate results as much as possible.

Requirements

- Platform: OS X or Linux
- Compiler: rust >= 1.11.0 + cargo (http://www.rust-lang.org)
- Genome tools (http://genometools.org)
- wigToBigWig (http://genome.ucsc.edu/admin/git.html or http://hgdownload.cse.ucsc.edu/admin/exe/)

seqOutBias is written in Rust, so it requires Rust and Cargo to compile it (www.rust-lang.org). *seqOutBias* is known to run on OS X and Linux. It may be possible to compile it on other platforms as long as supporting Rust libraries can be made to compile, specifically https://crates.io/crates/flate2 and https://crates.io/crates/rust-htslib are likely to be the dependencies that are most troublesome to compile.

At runtime, the step that computes mappability requires Genome Tools' Tallymer program to be in the PATH. It is possible to replicate the steps taken by *seqOutBias* to compute the mappability file independently, or run that part in a different machine, and supply the resulting file to *seqOutBias* (see the `tallymer` sub-command for more details).

The step that generates a bigWig file requires the UCSC executable `wigToBigWig` in the PATH.

Setup

Compilation

To install Rust and Cargo visit www.rust-lang.org, *seqOutBias* should compile with Rust 1.11.0 or later. Alternatively, on OS X, if you have Homebrew (http://brew.sh/) installed and up to date, you can use brew to install rust as so:

```
brew install rust
```

Afterwards, uncompress the source and build:

```
tar xzf guertinlab-seqOutBias-v1.1.0.tar.gz

cd guertinlab-seqOutBias-v1.1.0

cargo build --release
```

Installation

After compilation, copy the *seqOutBias* binary (`seqOutBias_1.1.0/target/release/seqOutBias`) to a folder in your PATH, for example `/usr/local/bin`.

You'll also need to install genome tools (http://genometools.org). Again, on OS X, if you have Homebrew (http://brew.sh) installed you can install genome tools as so:

```
brew install homebrew/science/genometools
```

You will need to build the wigToBigWig utility from the UCSC Genome Browser source code (http://genome.ucsc.edu/admin/git.html) and move the binary into your PATH, following the instructions included with the Genome Browser source code. This command is required to obtain bigWig files after scaling. Alternatively, you can download the platform-specific binary from UCSC (http://hgdownload.cse.ucsc.edu/admin/exe/).

Quick Start

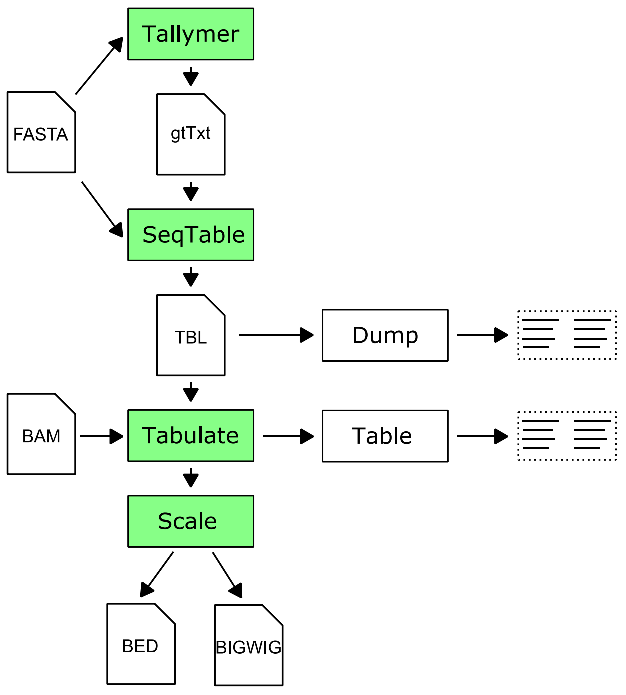
With *genometools, wigToBigWig* and *seqOutBias* installed and in the PATH, we can start processing files by simply invoking the command as:

```
seqOutBias ref.fa reads.bam
```

where **`ref.fa`** and **`reads.bam`** are, respectively, the reference sequence FASTA file and the **sorted** aligned reads BAM file[1]. This will run through the process of computing the mappability information, parsing the reference sequence to compute k-mer indexes, tallying the k-mer counts in both the sequence and the aligned reads, and finally producing the scaled BED and bigWig read pile-ups. Each intermediate step corresponds to a *seqOutBias* subcommand, allowing them to be run individually and even on different machines (see the next sections for more details).

When ran with no arguments, this will assume default values for read length (36), k-mer size (4) and the cut-site position (middle of the k-mer, i.e., plus and minus offset = 2). Run `seqOutBias -h` to see the full set of options and subcommands.

Overview



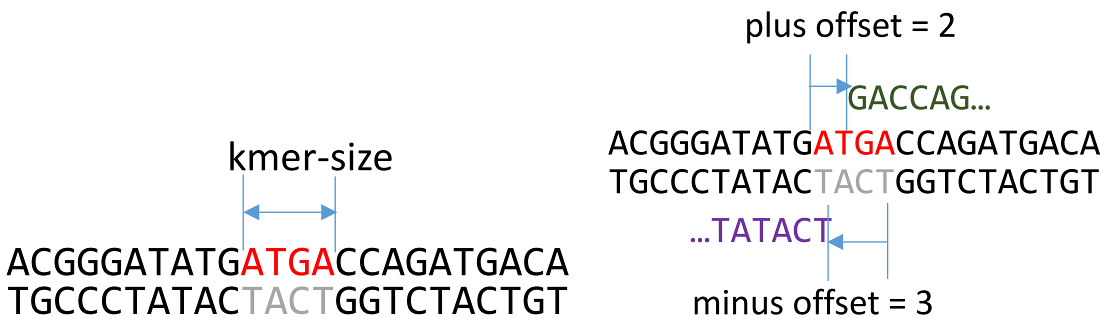
The figure above gives a high-level overview of the inputs (left side “documents”), intermediate files (mid-flow “documents”) and output (bottom) of the *seqOutBias* program. Furthermore, it illustrates the various computation steps that the program performs:

[1] It’s possible to supply more than one BAM file, for example, to combine the data from multiple replicates.

- Tallymer – Indexes the reference sequence (FASTA) and computes mappability for the given read length;
- SeqTable – Parses the reference sequence (FASTA) together with the mappability information to compute the k-mer that corresponds to each possible read alignment position; the resulting binary file stores this information in a compressed form that's easy to use for subsequent computation steps, as well as storing the corresponding parameters (read length, k-mer size, and cut-site offsets);
- Tabulate – Tallies the k-mer counts across the selected regions (or full genome), as well as the k-mers corresponding to observed aligned reads (if a BAM file is supplied);
- Scale – Compute the genome-wide aligned read pile-ups, scaling them by the expected/observed cut frequency;

When executed as described in the "Quick Start" section, all the main computation steps (marked in green) are executed, however, different subcommands can be used to run specific computation steps (see "Subcommands" section), or obtain text versions of the intermediate states ("dump" to display the SeqTable output, "Table" to obtain the normalization table data).

kmer-size definition



In *seqOutBias*, the sequence recognized by the enzyme to confer specificity, the k-mer, is characterized by four parameters (illustrated above): kmer-size, read size, and a pair of offsets for the plus and minus strands. This enables the use of *seqOutBias* with enzymes that have distinct recognition site lengths.

Note: Only non-negative value offsets are permitted.

Spaced k-mers

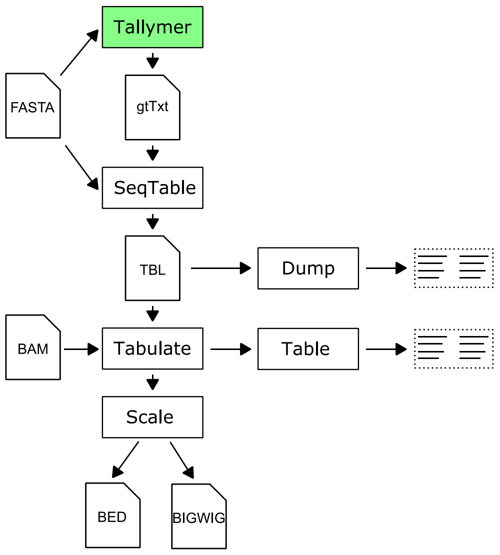
For situations where some bases surrounding the first sequenced base do not contribute to site recognition, it is possible to specify a kmer-mask, using the "`--kmer-mask`" parameter. Positions that should be ignored are represented by an 'X' and regular positions by an 'N'. For example, a possible 8-mer that spans 16 bp could be represented as NNXXNNXXXXNNXXNN.

This parameter also provides an alternative way to specify the position intervening between the base directly upstream the first base sequenced and the first base sequence by inserting a 'C' in the mask string. For example, "NNCNN", would represent a recognition site with size equal to four, plus-offset = 2 and minus-offset = 2.

Note: When the kmer-mask is present, it takes precedence over the "`--kmer-size`" parameter. If a 'C' is used to indicate the cut-site position, it takes precedence over the offset parameters.

Subcommands

tallymer



```
seqOutBias tallymer <fasta-file> <read-size> [--parts=<n>]
```

This subcommand creates the mappability file for a given read length. This process consists of three parts: 1) creating a suffix tree; 2) creating a genome index; 3) creating the mappability file.

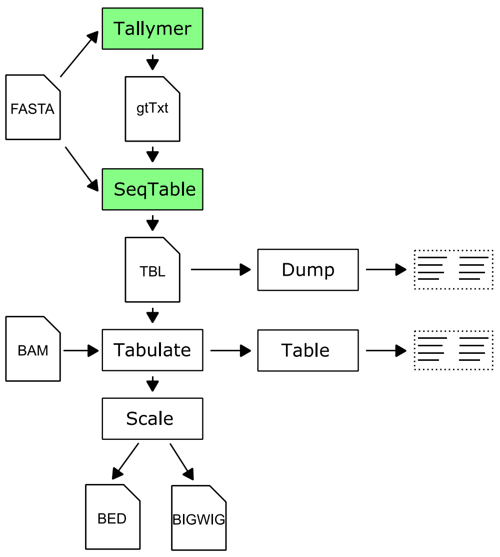
It corresponds to running the following genome tools' commands (replacing the bolded parts with the *seqOutBias* arguments):

```
gt suffixerator -dna -pl -tis -suf -lcp -v -parts <n> -db <fasta-file>
-indexname <fasta-file>.sft

gt tallymer mkindex -mersize <read-size> -minocc 2 -indexname <fasta-
file>.tal_<read-size> -counts -pl -esa <fasta-file>.sft

gt "tallymer search -output qseqnum qpos -strand fp -tyr <fasta-
file>.tal_<read-size> -q <fasta-file> > <fasta-file>.tal_<read-
size>.gtTxt
```

Intermediate files are created in the current working directory. Furthermore, *seqOutBias* will recognize the existence of intermediate files and avoid recomputing those portions. So, if seqOutBias tallymer is executed for different read sizes, but keeping the same FASTA file, then the first suffix-tree portion is re-used across invocations.

If the input FASTA file is compressed, then so will be the output file.

seqtable



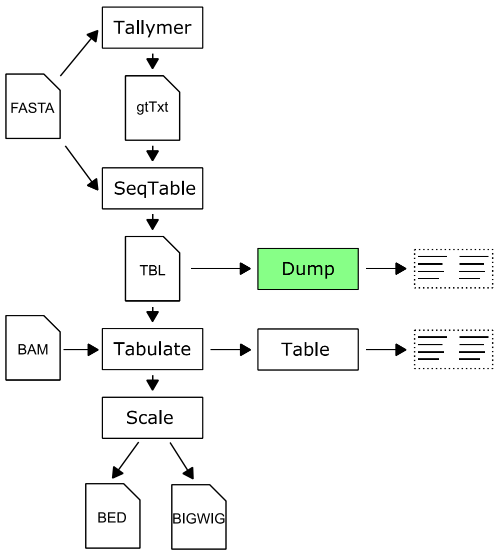
```
seqOutBias seqtable <fasta-file> [options]

--kmer-size=<n>     kmer size [default: 4].

--tallymer=<file>   Unmappable positions file produced by tallymer (seq, pos).

--plus-offset=<p>   Offset on plus strand [default: 2]. Eg, p=2 AA[A]A.

--minus-offset=<m>  Offset on minus strand [default: 2]. Eg, m=2 A[A]AA.

--kmer-mask=<str>   String indicating relevant kmer positions, eg.
NNXXNNCXXXXNNXXNN.

--read-size=<r>     Read length [default: 36].

--parts=<n>         Split suffix tree generation into n parts [default: 4].

--out=<outfile>      Output seqtable filename (defaults to fasta file basename with
.tbl extension).
```

This subcommand creates an intermediate table that combines mappability, read length, and offsets to map k-mer indexes to the aligned read positions. This intermediate file reduces the amount of computation needed when processing aligned read files and provides an intermediate file that decouples the reference sequence processing from the remaining steps.

Note: If no tallymer output file is supplied (via the `--tallymer=<file>` option), then it will invoke the tallymer subcommand.

Note: When the k-mer mask is present, it takes precedence over the "`--kmer-size`" parameter. If a 'C' is used in the "`--kmer-mask`" parameter to indicate the position relative to the first base sequenced, it takes precedence over the offset parameters.

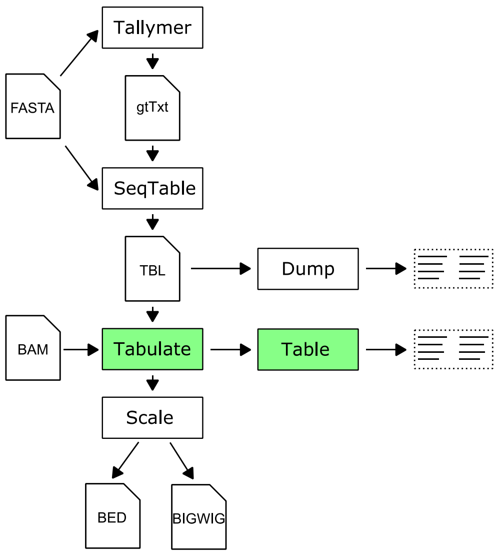
dump



```
seqOutBias dump <seqtbl-file> [<seqrange>]
```

This subcommand enables the inspection the seqtable file (or parts of it) by outputting the file contents in plain text. It takes an optional sequence range in the form of "`chrom:start-end`" that restricts the output to that region. The output is preceded by a list of the input parameters (read-size, kmer-size, etc.) that were used to build the sequence table file.

table



```
seqOutBias    table    <seqtbl-file>    [<bam-file>...]    [--qual=<q>]    [--regions=<bedfile>] [--pdist=<min:max>] [--only-paired]
```

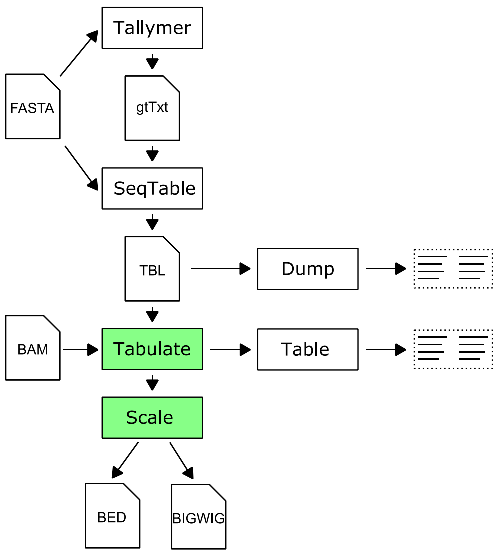
This subcommand produces an k-mer count table based on the sequence information (via the seqtbl file) and the optional BAM files. If more than one file is given, then the counts are pooled across files. **K-mers are indexed by their plus strand sequence** (for example, AAAA = 1, AAAC = 2, AAAG = 3, etc), but counts are reported independently for each strand. It's done this way because mappability can differ between strands. By default counts correspond to the entire genome, but can be constrained to specific regions by supplying a BED file with the "--regions" option.

When no BAM file is supplied, the output will have four columns: k-mer index, k-mer string, plus strand count, and minus strand count. If a BAM file is supplied, the output will have two additional columns with the plus and minus strand counts of observed aligned reads. For example:

```
1     AAAA  40584084    39080470
2     AAAC  14531729    14684467
3     AAAG  19041196    19266837
4     AAAT  26793796    27032258
5     AACA  15881340    16097718
6     AACC  7943846     8098141
7     AACG  1562521     1571244
8     AACT  12186297    12417246
9     AAGA  17771087    18024000
...
```

If the `--exact-length` flag was set, aligned reads are filtered by read length; only the reads that match the read length will be used to produce the sequence table are used. The user can optionally filter quality score of the aligned reads. For paired reads, it is also possible to filter by their distance (furthest edge-to-edge) and to discard singleton reads. Unaligned reads are discarded.

scale



```
seqOutBias scale <seqtbl-file> <bam-file>... [options]
```

```
  --qual=<q>             Minimum read quality [default: 0].

  --regions=<bedfile>    Count only cut-sites inside the regions indicated in the BED
file.

  --bed=<bedfile>         Output scaled BED filename (defaults to BAM file basename
with '_scaled.bed' extension).

  --skip-bed             Skip creating the BED file output.

  --bw=<bigwigfile>      Output scaled bigWig filename (defaults to BAM file basename
with .bw extension).

  --skip-bw              Skip creating the bigWig file output.

  --stranded             Output per strand counts when writing scaled values.

  --shift-counts         Shift minus strand counts.

  --no-scale             Skip actual scaling in 'scale' command.

  --pdist=<min:max>      Distance range for included paired reads.

  --only-paired          Only accept aligned reads that have a mapped pair.

  --tail-edge            Use tail edge of reads (3') instead of start edge (5').
```

This subcommand produces the scaled aligned read pile-ups, both in BED and bigWig form. Listed above are the options to control the output. Aside from the filtering options that apply to the "tabulate" computation, there are a few options to tweak the results, namely:

- Scaling can be turned off using the "--no-scale" option;
- Minus strand pile-up position can be shifted to align with the plus strand pile-up ("--shift-counts" option), making reads from both sides of a cleavage site count to the same position;

- Output per strand, using the “--stranded" option will result in one bigWig file per strand, as well as per strand counts inside the BED file (plus strand with a positive sign, minus strand with a negative sign);
- Output type selection, using either the “--skip-bed” or the “--skip-bw” option will omit the production of the BED or bigWig file respectively.
- For PRO-seq and similar protocols, the “--tail-edge” option will use the 3’ end of the read.
- Output file name, using the “--bed" or “--bw" flags;

seqOutBias API

The code is structured into two parts: a main library (seqoutbiaslib) and the command line program (*seqOutBias*) implemented using that library. This split allows the code to be reused to implement different interfaces with similar functionality or as a component in a larger program. We expose the library as both a Rust library and a C library.

Using from Rust

The Rust interface is the most extensive, since it includes everything used to build the main program. The code is split into a series of modules which roughly correspond to the different subcommands of *seqOutBias*:

- tallyrun – code to execute genome tools to produce the mappability file
- tallyread – code to read and access the mappability information
- seqtable – code to read and write *seqtable* files to disk
- fasta – code to read in the FASTA file, combine it with the mappability information, and produce the seqtable file (via calls to the seqtable module)
- filter – code to filter BAM records based on things like length, quality, etc.
- counts – code to tabulate kmer counts
- scale – code to compute read pile-ups and scale them appropriately
- bigwig – code to write the chromInfo and wiggle files and convert them to a bigWig file via the wigToBigWig program

To use the library from a new rust program, simply add the following to your Cargo.toml file:

```
[dependencies]

seqOutBias = { git = "https://github.com/guertinlab/seqOutBias" }
```

and add the appropriate declarations to your code. For example (see also src/main.rs):

```
extern crate seqoutbiaslib;

use seqoutbiaslib::tallyrun;

use seqoutbiaslib::seqtable;

use seqoutbiaslib::fasta;

use seqoutbiaslib::counts;

use seqoutbiaslib::scale;

use seqoutbiaslib::file_exists;
```

Using from C

The C API is more limited, exposing the ability to generate a seqtable file and the ability to create and query pile-ups in memory (without writing them to disk). The C library (in the form of a .so file in Linux or a .dylib file on OS X) and the corresponding header file (seqoutbias.h) are built as part of the main compilation process and can be linked in as usual for regular C libraries.

Functions can be grouped into four sets:

1) Functions to manage the seqtable generation parameters (the SeqTblParams structure is opaque):

```
SeqTblParams* seqoutbias_params_with_mask(char const* kmer_mask,
uint8_t plus_offset, uint8_t minus_offset, uint16_t read_length);


SeqTblParams* seqoutbias_params(uint8_t kmer_size, uint8_t plus_offset,
uint8_t minus_offset, uint16_t read_length);


void seqoutbias_free_params(SeqTblParams* ptr);
```

2) A function to create a default set of pile-up generation parameters (the Config structure is user public):

```
Config seqoutbias_default_config(void);
```

3) A function to generate the seqtable file:

```
int32_t seqoutbias_generate_seqtbl(char const* fasta_filename,
SeqTblParams* params, char const* output_filename);
```

4) Functions to generate and query the pile-ups (the PileUpDate structure is opaque):

```
PileUpData* seqoutbias_create_pileup(char const* seqtable_filename,
char const* const* bam_filenames, size_t n_bams, Config config);


void seqoutbias_free_pileup(PileUpData* ptr);


PileUpPoint* seqoutbias_chrom_pileup(PileUpData* ptr, char const*
chrom, size_t* out_len);


int32_t seqoutbias_query_pileup_chrom_index(PileUpData* ptr, char
const* chrom, size_t* out_index);


int32_t seqoutbias_query_pileup(PileUpData* ptr, size_t chrom_index,
uint32_t pos, PileUpPoint* out);
```

Please see the auto-generated include file (target/include/seqoutbias.h) for details.