ConsensusCruncher

Release 0.0.2

Aug 30, 2019

Contents:

Welcome to the ConsensusCruncher documentation!

ConsensusCruncher is a tool that suppresses errors in next-generation sequencing data by using unique molecular identifiers (UMIs) to amalgamate reads derived from the same DNA template into a consensus sequence.

To learn more about ConsensusCruncher and its applications, see our publication in Nucleic Acids Research.

CHAPTER 1

Quick start guide

1.1 Installation

1. Download the latest release or clone the repository:

```
$ git clone https://github.com/pughlab/ConsensusCruncher.git
```

2. Install the dependencies:

Program	Version	Purpose
Python	3.5.1	Run ConsensusCruncher
BWA	0.7.15	Align reads
Samtools	1.3.1	Sorting and indexing bamfiles

3. All required python libraries can be installed by running:

```
$ pip install -r requirements.txt
```

1.2 Configuration

Set up `config.ini` with the appropriate configurations for `fastq2bam` and `consensus` modes. Alternatively, you can provide command-line arguments, which will overwrite `config.ini` parameters.

Example config file:

```
[fastq2bam]
fastq1 = # Path to FASTQ containing Read 1 of paired-end reads. [MANDATORY]
fastq2 = # Path to FASTQ containing Read 2 of paired-end reads. [MANDATORY]
output = # Output directory, where barcode extracted FASTQ and
         # BAM files will be placed in subdirectories 'fastq_tag'
         # and 'bamfiles' respectively (dir will be created if
```

(continues on next page)

(continued from previous page)

```
         # they do not exist). [MANDATORY]
name = # Sample name extracted from filename using read number delimiter
         # (e.g. Sample1_R1.fastq, delimiter = '_R', sample name = "Sample1").
↪[MANDATORY]
bwa = # Path to executable BWA. [MANDATORY]
ref = # Path to reference (BWA index). [MANDATORY]
samtools = # Path to executable samtools. [MANDATORY]
bpattern = # Barcode pattern (N = random barcode bases, A|C|G|T = fixed spacer bases).
↪ [MANDATORY]
blist = # List of barcodes (Text file with unique barcodes on each line). [MANDATORY]

# Note: You can input either a barcode list or barcode pattern or both.
# If both are provided, barcodes will first be matched with the list and then the
↪constant
# spacer bases will be removed before the barcode is added to the header.
# e.g. ATNNGT means 2-bp barcode is flanked by two spacers matching 'AT' in front and
# 'GT' behind.

[consensus]
bam  = # Input BAM file with barcodes in header. [MANDATORY]
c_output = # Output directory for consensus sequences. [MANDATORY]
samtools = # Path to executable samtools. [MANDATORY]

# Optional arguments
scorrect = # Singleton correction [True or False, default: True].
genome = # Genome version to determine which cytoband file to use for data splitting
         # (hg19 or hg38, default: hg19)
bedfile = # Bedfile, default: cytoBand.txt. WARNING: It is HIGHLY RECOMMENDED that you
          # use the provided cytoBand.txt unless you're working with genome build that
          # is not hg19 or hg38. Then a separate bedfile is needed for data
↪segmentation.
          # For small BAM files, you may choose to turn off data splitting with '-b
↪False'
          # and process everything all at once (Division of data is only required for
          # large data sets to offload the memory burden).
cutoff = # Consensus cut-off, default: 0.7
         # (70% of reads must have the same base to form a consensus).
bdelim = # Delimiter before barcode in read name
         # (e.g. '|' in 'HWI-D00331:196:C900FANXX:7:1110:14056:43945|TTTT')
cleanup = # Remove intermediate files. [True or False]
```

UMI list example:

```
TATGCGT
AACGGAT
AAGCCT
AATCGCT
ACCGAT
ACGCAAT
ACGTGT
AGGACAT
ATGTCCT
CACAGT
```

1.3 Running ConsensusCruncher

ConsensusCruncher has 2 modes:

1.3.1 Extract UMIs & align FASTQs

1. **`fastq2bam` extracts UMIs and aligns FASTQs with BWA to generate BAM files.** Run `ConsensusCruncher.py [-c CONFIG] fastq2bam` with required input parameters (see `config.ini` or `--help` for examples).

   This script extracts molecular barcode tags and removes spacers from unzipped FASTQ files found in the input directory (file names must contain "R1" or "R2"). Barcode extracted FASTQ files are written to the 'fastq_tag' directory and are subsequently aligned with BWA mem. Bamfiles are written to the 'bamfile" directory under the project folder.

1.3.2 Error suppression

2. **`consensus` generates single-strand and duplex consensus sequences with 'Singleton Correction'** Run `ConsensusCruncher.py [-c CONFIG] consensus` with the required input parameters (see `config.ini` or `--help` for examples).

   Using unique molecular identifiers (UMIs), duplicate reads from the same molecule are amalgamated into single-strand consensus sequences (SSCS). If complementary strands are present, SSCSs can be subsequently combined to form duplex consensus sequences (DCS).

   If 'Singleton Correction' (SC) is enabled, single reads (singletons) can be error suppressed using complementary strand. These corrected singletons can be merged with SSCSs to be further collapsed into DCSs + SC.

   Finally, a BAM file containing only unique molecules (i.e. no duplicates) is created by merging DCSs, remaining SSCSs (those that could not form DCSs), and remaining singletons (those that could not be corrected).

1.4 Multiple FASTQs

`ConsensusCruncher.py` processes one sample (2 paired-end FASTQ files or 1 BAM file) at a time. A sample script to generate shell scripts for multiple samples is provided here.

The script generator will create sh scripts for each file in a fastq directory.

1. The following parameters need to be changed in the config file: name, bwa, ref, samtools, bpattern (alternatively if a barcode list is used instead, remove bpattern and add blist as parameter). Please note: fastq1, fastq2, output, bam, and c_output can be ignored as those will be updated using the generate_scripts.sh file.
2. Update generate_scripts.sh with input, output, and code_dir.
3. Run generate_scripts.sh to create sh files and then run those scripts.

1.5 Output

Please note the example below is for illustrative purposes only, as sample names and index files were removed for simplification. Order of directories and files were also altered to improve comprehension.

```
.                                                Filetype
├── sscs
│   ├── badReads.bam                             Reads that are unmapped or have multiple␣
↪alignments
│   ├── sscs.sorted.bam                          Single-Strand Consensus Sequences (SSCS)
│   ├── singleton.sorted.bam                     Single reads (Singleton) that cannot form␣
↪SSCSs
├── sscs_SC
|   ├── singleton.rescue.sorted.bam              Singleton correction (SC) with␣
↪complementary singletons
|   ├── sscs.rescue.sorted.bam                   SC with complementary SSCSs
|   ├── sscs.sc.sorted.bam                       SSCS combined with corrected singletons␣
↪(from both rescue strategies)   [*]
|   ├── rescue.remaining.sorted.bam              Singletons that could not be corrected
|   ├── all.unique.sscs.sorted.bam               SSCS + SC + remaining (uncorrected)␣
↪singletons
├── dcs
│   ├── dcs.sorted.bam                           Duplex Consensus Sequence (DCS)
│   ├── sscs.singleton.sorted.bam                SSCSs that could not form DCSs as␣
↪complementary strand was missing
├── dcs_SC
│   ├── dcs.sc.sorted.bam                        DCS generated from SSCS + SC    [*]
│   ├── sscs.sc.singleton.sorted.bam             SSCS + SC that could not form DCSs
│   ├── all.unique.dcs.sorted.bam                DCS (from SSCS + SC) + SSCS_SC_Singletons␣
↪+ remaining singletons
├── read_families.txt                            Family size and frequency
├── stats.txt                                    Consensus sequence formation metrics
├── tag_fam_size.png                             Distribution of reads across family size
└── time_tracker.txt                             Time log
```

Through each stage of consensus formation, duplicate reads are collapsed together and single reads are written as separate files. This allows rentention of all unique molecules, while providing users with easy data management for cross-comparisons between error suppression strategies.

To simplify analyses, it would be good to focus on SSCS+SC ("sscs.sc.sorted.bam") and DCS+SC ("dcs.sc.sorted.bam") as highlighted above with [*].

CHAPTER 2

Tutorial

Sample FASTQ files can be found under the test folder. Please note these FASTQ are only for testing purposes. For the full FASTQs used in our paper, please download the data from the NCBI Sequence Read Archive (SRA; https://www.ncbi.nlm.nih.gov/sra/) under access numbers SRP140497 and SRP141184.

In order to create consensus sequences, we first need to process FASTQ files into BAM files.

2.1 FASTQs to BAMs

Given FASTQs as input files, `fastq2bam` mode removes the spacer region and extracts the barcode tag from each sequencing read into the header with extract_barcode.py. The tag removed FASTQs are then aligned with BWA mem into BAM files (Arguments can be provided in the *config.ini* file or as command-line arguments. Please note command-line arguments over-writes config.ini arguments).

```
REPO="[insert path to ConsensusCruncher repo]"
BWAPATH="[insert path to BWA]"
BWAINDEX="[insert path to BWA INDEX]"
BWAPATH="[insert path to SAMTOOLS]"

python ConsensusCruncher.py fastq2bam --fastq1 $REPO/test/fastq/LargeMid_56_L005_R1.
↪fastq
--FASTQ2 $REPO/test/fastq/LargeMid_56_L005_R2.fastq -o $REPO/test -b $BWAPATH -r
↪$BWAIndex
-s $SAMTOOLS -bpattern NNT
```

In the sample dataset, we utilized 2-bp (NN) barcodes and 1-bp (T) spacers. While the barcodes for each read can be one of 16 possible combinations (4^2), the spacer is an invariant "T" base used to ligate barcodes onto each end of a DNA fragment. Thus, a spacer filter is imposed to remove faulty reads. Barcodes from read 1 and read 2 are extracted and combined together before being added to the header.

```
READ FROM SEQUENCER
Read1:
@HWI-D00331:196:C900FANXX:5:1101:1332:2193 1:N:0:ACGTCACA    [<-- HEADER]
```

(continues on next page)

(continued from previous page)

```
ATTAAGCCCCAGGCAGTTGCTAATGATGGGAGCTTAGTGCACAAGGGCTGGGCCTCCCTCTTGGAGCTGAACATTGTTTCTTGGGGACGGCTGTGCCCACC
↪ [<-- SEQUENCE]
+
BCCCCGGGGGGGGGGGGGGGGGGGGGGGGFGGGGGGGGEGGGGGBGGGGGGGGGGGGGGGGGGGGGGEGG1:FGFGGGGGGGGG/
↪CB>DG@GGGGGGG<DGGGGAAGGEGGB>DGGGEGGG/@G  [<-- QUALITY SCORE]

Read2:
@HWI-D00331:196:C900FANXX:5:1101:1332:2193 2:N:0:ACGTCACA
GGTGGGCTCCAGCCCTGATTTCCTCCCCCAGCCCTGCAGGGCTCAGGTCCAGAGGACACAAGTTTAACTTGCGGGTGGTCACTTGCCTCGTGCGGTGACGC
+
BBBBCGGGGEGGGGFGGGGGGGGGGGGGGGGGGGGGGB:FCGGGGGGGGGGEGGGGGGGG=FCGG:@GGGEGBGGGAGFGDE@FGGGGGFGFGEGDGGGFC
↪#

------

AFTER BARCODE EXTRACTION AND SPACER ("T") REMOVAL
Read1:
@HWI-D00331:196:C900FANXX:5:1101:1332:2193|AT.GG/1
AAGCCCCAGGCAGTTGCTAATGATGGGAGCTTAGTGCACAAGGGCTGGGCCTCCCTCTTGGAGCTGAACATTGTTTCTTGGGGACGGCTGTGCCCACCTCA
+
CCGGGGGGGGGGGGGGGGGGGGGGGGFGGGGGGGGEGGGGGBGGGGGGGGGGGGGGGGGGGGGGEGG1:FGFGGGGGGGGG/
↪CB>DG@GGGGGGG<DGGGGAAGGEGGB>DGGGEGGG/@G

Read2:
@HWI-D00331:196:C900FANXX:5:1101:1332:2193|AT.GG/2
GGGCTCCAGCCCTGATTTCCTCCCCCAGCCCTGCAGGGCTCAGGTCCAGAGGACACAAGTTTAACTTGCGGGTGGTCACTTGCCTCGTGCGGTGACGCCAT
+
BCGGGGEGGGGFGGGGGGGGGGGGGGGGGGGGGGB:FCGGGGGGGGGGEGGGGGGGG=FCGG:@GGGEGBGGGAGFGDE@FGGGGGFGFGEGDGGGFCGGI
↪#
```

FASTQ files with extracted barcodes are placed in the fastq_tag directory and are subsequently aligned with BWA to generate BAMs in the bamfiles folder.

```
.
├── bamfiles
├── fastq
├── fastq_tag
└── qsub
```

2.2 ConsensusCruncher

`consensus` mode creates a consensus directory and folders for each bam file. BAM files undergo consensus construction through the workflow illustrated above. Output BAMs are grouped according to type of error suppression (SSCS vs DCS) and whether Singleton Correction (SC) was implemented.

```
.
├── bamfiles
├── consensus
│   ├── LargeMid_56_L005
│   │   ├── dcs
│   │   ├── dcs_SC
│   │   ├── sscs
│   │   └── sscs_SC
...
```

(continues on next page)

(continued from previous page)

```
│   ├── LargeMid_62_L006
│   │   ├── dcs
│   │   ├── dcs_SC
│   │   ├── sscs
│   │   └── sscs_SC
│   └── qsub
├── fastq
├── fastq_tag
└── qsub
```

Within a sample directory (e.g. LargeMid_56_L005), you will find the following files:

Please note the example below is for illustrative purposes only, as sample names and index files were removed for simplification. Order of directories and files were also altered to improve comprehension.

```
.                                           Filetype
├── sscs
│   ├── badReads.bam                        Reads that are unmapped or have multiple␣
↪alignments
│   ├── sscs.sorted.bam                     Single-Strand Consensus Sequences (SSCS)
│   ├── singleton.sorted.bam                Single reads (Singleton) that cannot form␣
↪SSCSs
├── sscs_SC
|   ├── singleton.rescue.sorted.bam         Singleton correction (SC) with␣
↪complementary singletons
|   ├── sscs.rescue.sorted.bam              SC with complementary SSCSs
|   ├── sscs.sc.sorted.bam                  SSCS combined with corrected singletons␣
↪(from both rescue strategies)   [*]
|   ├── rescue.remaining.sorted.bam         Singletons that could not be corrected
|   ├── all.unique.sscs.sorted.bam          SSCS + SC + remaining (uncorrected)␣
↪singletons
├── dcs
│   ├── dcs.sorted.bam                      Duplex Consensus Sequence (DCS)
│   ├── sscs.singleton.sorted.bam           SSCSs that could not form DCSs as␣
↪complementary strand was missing
├── dcs_SC
│   ├── dcs.sc.sorted.bam                   DCS generated from SSCS + SC    [*]
│   ├── sscs.sc.singleton.sorted.bam        SSCS + SC that could not form DCSs
│   ├── all.unique.dcs.sorted.bam           DCS (from SSCS + SC) + SSCS_SC_Singletons␣
↪+ remaining singletons
├── read_families.txt                       Family size and frequency
├── stats.txt                               Consensus sequence formation metrics
├── tag_fam_size.png                        Distribution of reads across family size
└── time_tracker.txt                        Time log
```

Through each stage of consensus formation, duplicate reads are collapsed together and single reads are written as separate files. This allows rentention of all unique molecules, while providing users with easy data management for cross-comparisons between error suppression strategies.

To simplify analyses, it would be good to focus on SSCS+SC ("sscs.sc.sorted.bam") and DCS+SC ("dcs.sc.sorted.bam") as highlighted above with [*].

Within the stats file you should expect to see the following (Please note as this is a test dataset, the number of consensus reads is very low):

```
# === SSCS ===
Uncollapsed - Total reads: 19563
```

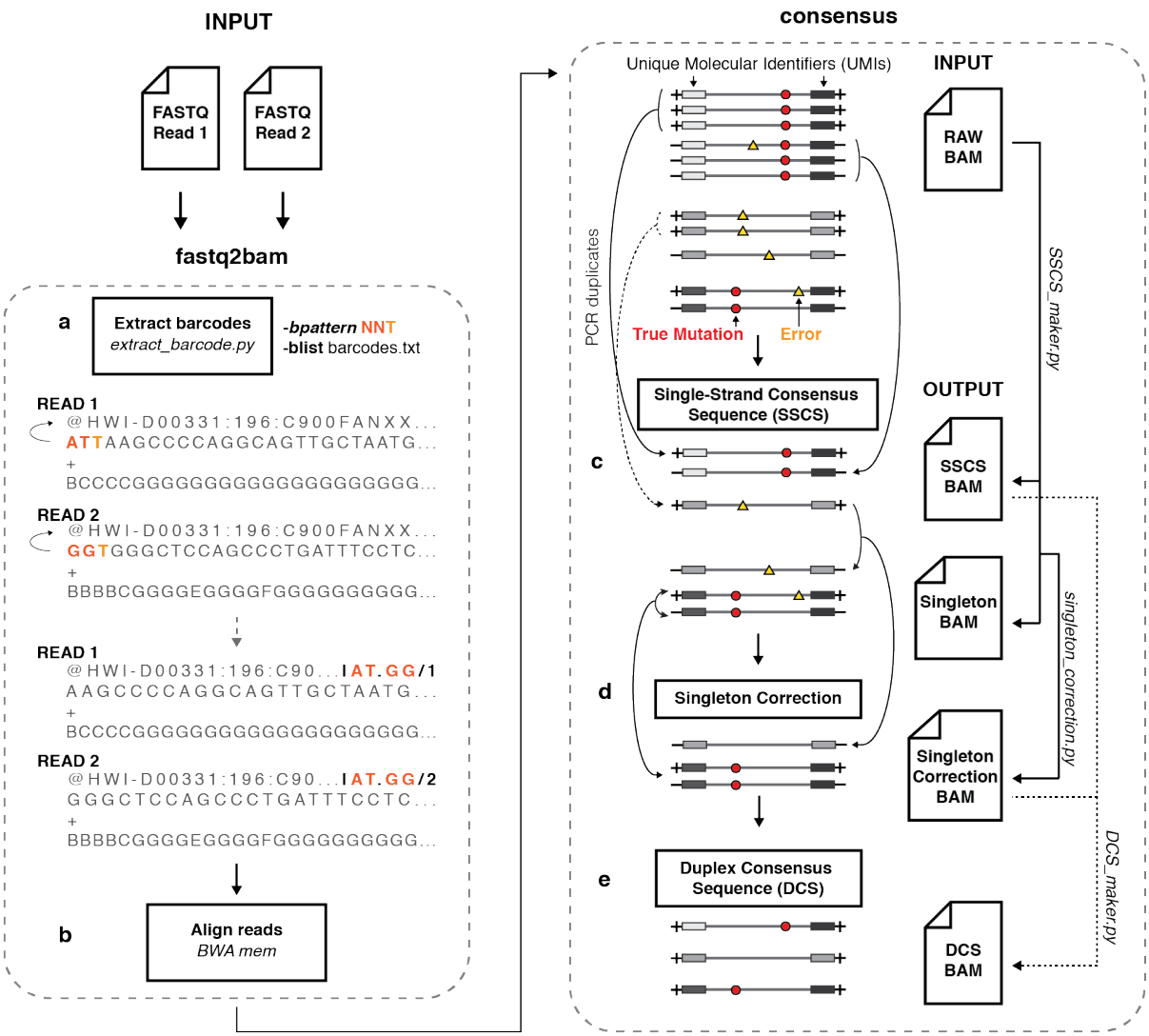
(continues on next page)

(continued from previous page)

```
Uncollapsed - Unmapped reads: 17
Uncollapsed - Secondary/Supplementary reads: 24
SSCS reads: 0
Singletons: 19522
Bad spacers: 0

# QC: Total uncollapsed reads should be equivalent to mapped reads in bam file.
Total uncollapsed reads: 19563
Total mapped reads in bam file: 19563
QC: check dictionaries to see if there are any remaining reads
=== pair_dict remaining ===
=== read_dict remaining ===
=== csn_pair_dict remaining ===
0.02919737100601196
# === DCS ===
SSCS - Total reads: 0
SSCS - Unmapped reads: 0
SSCS - Secondary/Supplementary reads: 0
DCS reads: 0
SSCS singletons: 0

# === Singleton Correction ===
Total singletons: 19522
Singleton Correction by SSCS: 0
% Singleton Correction by SSCS: 0.0
Singleton Correction by Singletons: 4
% Singleton Correction by Singletons : 0.020489703923778302
Uncorrected Singletons: 19518

0.020557292302449546
# === DCS - Singleton Correction ===
SSCS SC - Total reads: 4
SSCS SC - Unmapped reads: 0
SSCS SC - Secondary/Supplementary reads: 0
DCS SC reads: 2
SSCS SC singletons: 0
```

CHAPTER 3

How it works



Unique molecular identifiers (UMIs) composed of molecular barcodes and sequence features are used aggregate reads derived from the same strand of a template molecule. Amalgamation of such reads into single strand consensus sequences (SSCS) removes discordant bases, which effectively eliminates polymerase and sequencer errors. Complementary SSCSs can be subsequently combined to form a duplex consensus sequence (DCS), which eliminates asymmetric strand artefacts such as those that develop from oxidative damage.

Conventional UMI-based strategies rely on redundant sequencing from both template strands to form consensus sequences and cannot error suppress single reads (singleton). We enable singleton correction using complementary duplex reads in the absence of redundant sequencing.

ConsensusCruncher schematic:

- An uncollapsed bamfile is first processed through SSCS_maker.py to create an error-suppressed single-strand consensus sequence (SSCS) bamfile and an uncorrected singleton bamfile.
- The singletons can be corrected through singleton_correction.py, which error suppress singletons with its complementary SSCS or singleton read.
- SSCS reads can be directly made into duplex consensus sequences (DCS) or merged with corrected singletons to create an expanded pool of DCS reads (Figure illustrates singleton correction merged work flow).

CHAPTER 4

FAQ

- **Can I run ConsensusCruncher in parallel?** Yes, we provide `generate_scripts.sh` as a wrapper to create scripts for each individual sample. If you have access to a cluster, you can submit these scripts as separate jobs to be run in parallel.
- **Why is ConsensusCruncher taking so long?** The time it takes ConsensusCruncher to run depends on the sequencing depth, number of UMIs, and the computer you're using. Time can range from a few minutes to several hours (& up to a day with some of the larger samples). Please be patient!
- **Why is ConsensusCruncher taking so much memory?** The `consensus` mode of ConsensusCruncher groups reads of a BAM file by UMI and condenses it into consensus sequences. The amount of memory is dependent on the size of your BAM files. If you have large BAMs and have access to a cluster, we recommend running ConsensusCruncher on the high memory queue.
- **Why do we need a cytoband bedfile to separate the data?** Depending on the size of your BAM file, it may be too large to load all the reads into memory. We use coordinates of cytobands to split our data, but you could also use a target bedfile if you're only interested in specific regions of the genome.
- **Can I input BAMs into ConsensusCruncher?** Yes, as long as your BAM files have UMIs in the queryname of each read, it can be used for the `consensus` mode of ConsensusCruncher. (Please note: UMIs need to be separated by '|' and paired UMIs split by '.'). e.g. "HWI-D00331:196:C900FANXX:5:1101:11551:2948|AC.CT"
- **Can I use ConsensusCruncher for single-end sequencing data?** Yes, you can use the `SSCS_maker.py` directly to generate consensus sequences. However, please note that single-end sequencing data won't benefit from Singleton Correction as that requires paired-end data.

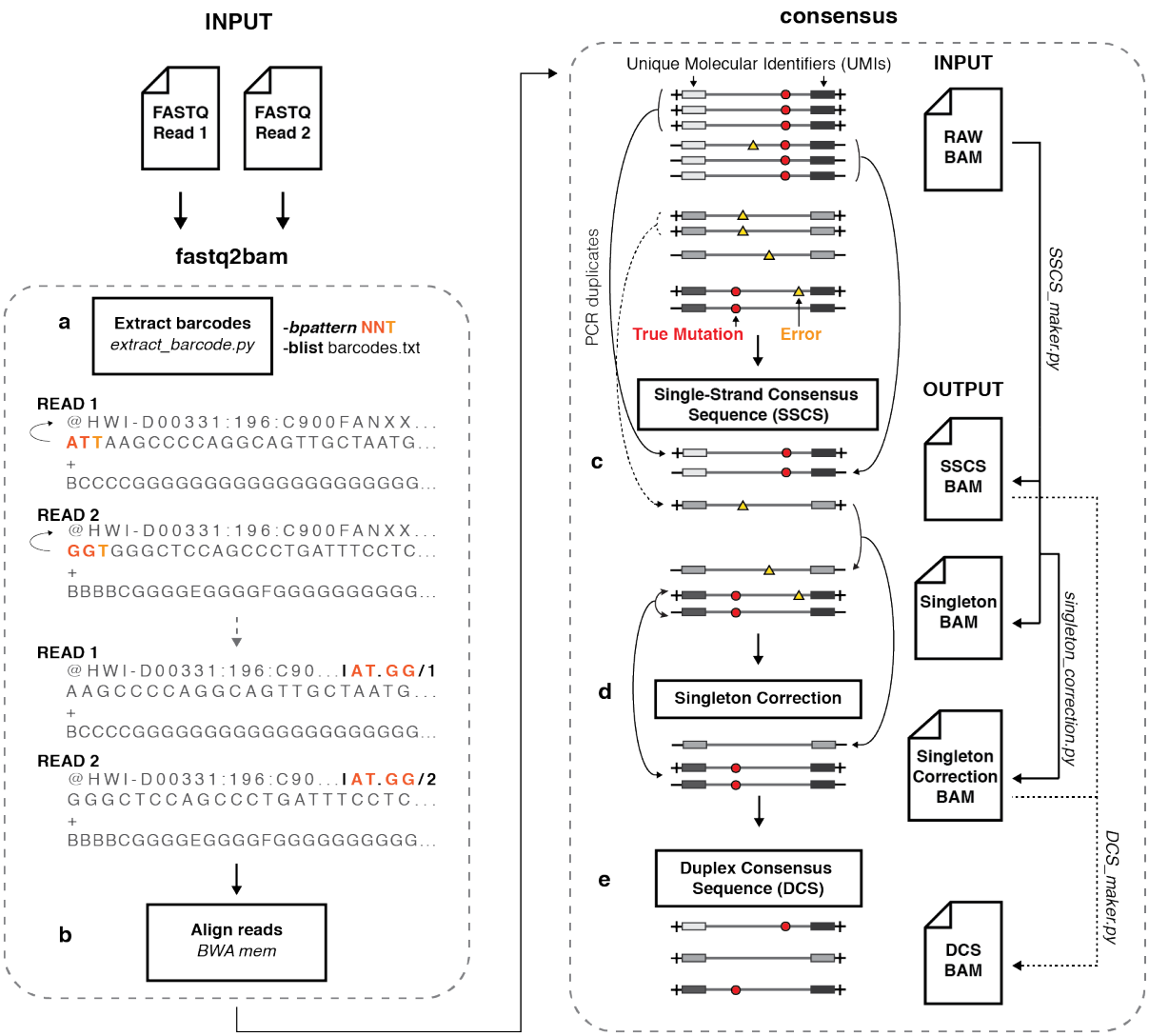
4.1 Questions

If you have any questions, please post it as an issue on our GitHub.

Alternatively, you can contact: Nina Wang (nina.tt.wang@gmail.com), Trevor Pugh (Trevor.Pugh@uhn.ca), Scott Bratman (Scott.Bratman@rmp.uhn.ca)

CHAPTER 5

Modes



ConsensusCruncher has 2 modes:

- **`fastq2bam` extracts UMIs and aligns FASTQs with BWA to generate BAM files.** This script extracts molecular barcode tags and removes spacers from unzipped FASTQ files found in the input directory (file names must contain "R1" or "R2"). Barcode extracted FASTQ files are written to the 'fastq_tag' directory and are subsequently aligned with BWA mem. Bamfiles are written to the 'bamfile" directory under the project folder.
- **`consensus` generates single-strand and duplex consensus sequences with 'Singleton Correction'** This script amalgamates duplicate reads in bamfiles into single-strand consensus sequences (SSCS), which are subsequently combined into duplex consensus sequences (DCS). Singletons (reads lacking duplicate sequences) are corrected, combined with SSCS to form SSCS + SC, and further collapsed to form DCS + SC. Finally, files containing all unique molecules (a.k.a. no duplicates) are created for SSCS and DCS.

5.1 Extract barcodes

`extract_barcodes.py`

Function: To isolate duplex barcodes from paired-end sequence reads and store in FASTQ headers after removal of spacer regions.

(Written for Python 3.5.1)

USAGE: python3 extract_barcodes.py [–read1 READ1] [–read2 READ2] [–outfile OUTFILE] [–blen BARCODELEN] [–slen SPACERLEN] [–sfilt SPACERFILT]

Arguments:

–read1 READ1	Input FASTQ file for Read 1 (unzipped)
–read2 READ2	Input FASTQ file for Read 2 (unzipped)
–outfile OUTFILE	Output FASTQ files for Read 1 and Read 2 using given filename
–bpattern BPATTERN	Barcode pattern (N = random barcode bases, A|C|G|T = fixed spacer bases)
–blist BARCODELIST	List of correct barcodes

Barcode design: N = random / barcode bases

A | C | G | T = Fixed spacer bases

e.g. ATNNGT means barcode is flanked by two spacers matching 'AT' in front and 'GT' behind

Inputs:

1. A FASTQ file containing first-in-pair (Read 1) reads
2. A FASTQ file containing second-in-pair (Read 2) reads

Outputs:

1. A Read 1 FASTQ file with barcodes added to the FASTQ header
2. A Read 2 FASTQ file with barcodes added to the FASTQ header
3. A text file summarizing barcode stats

5.2 Single-strand consensus sequences (SSCS)

`SSCS_maker.py`

Function: To generate single strand consensus sequences for strand based error suppression.

- Consensus sequence from most common base with quality score >= Q30 and greater than <cutoff> representation
- Consensus quality score from addition of quality scores (i.e. product of error probabilities)

(Written for Python 3.5.1)

Usage: python3 SSCS_maker.py [–cutoff CUTOFF] [–infile INFILE] [–outfile OUTFILE] [–bedfile BEDFILE]

Arguments:

–cutoff CUTOFF	**Proportion of nucleotides at a given position in a sequence required to** • Recommendation: 0.7 based on previous literature Kennedy et al. • **Example (–cutoff = 0.7) - four reads (readlength = 10) are as** – Read 1: ACTGATACTT – Read 2: ACTGAAACCT – Read 3: ACTGATACCT – Read 4: ACTGATACTT • The resulting SSCS is: ACTGATACNT
–infile INFILE	Input BAM file
–outfile OUTFILE	Output BAM file
–bedfile BEDFILE	Bedfile containing coordinates to subdivide the BAM file (Recommendation: cytoband.txt)

Inputs:

1. A position-sorted BAM file containing paired-end reads with duplex barcode in the header
2. A BED file containing coordinates subdividing the entire ref genome for more manageable data processing

Outputs:

1. A SSCS BAM file containing paired single stranded consensus sequences - "sscs.bam"
2. A singleton BAM file containing single reads - "singleton.bam"
3. A bad read BAM file containing unpaired, unmapped, and multiple mapping reads - "badReads.bam"
4. A text file containing summary statistics (Total reads, Unmmaped reads, Secondary/Supplementary reads, SSCS reads, and singletons) - "stats.txt"
5. A tag family size distribution plot (x-axis: family size, y-axis: number of reads) - "tag_fam_size.png"
6. A text file tracking the time to complete each genomic region (based on bed file) - "time_tracker.txt"

Concepts:

- Read family: reads that share the same molecular barcode, genome coordinates for Read1 and Read2, cigar string, strand, flag, and read number
- Singleton: a read family containing only one member (a single read)

5.3 Singleton Correction

`singleton_correction.py`

Function: To correct single reads with its complementary (SSCS/singleton) strand and enable error suppression

- Traditionally, consensus sequences can only be made from 2 or more reads

(Written for Python 3.5.1)

Usage: Python3 singleton_correction.py [–singleton Singleton BAM] [–bedfile BEDFILE]

Arguments:

–singleton Singleton-BAM	input singleton BAM file
–bedfile BEDFILE	Bedfile containing coordinates to subdivide the BAM file (Recommendation: cytoband.txt)

Inputs:

1. A position-sorted BAM file containing paired-end single reads with barcode identifiers in the header/query name
2. A BED file containing coordinates subdividing the entire ref genome for more manageable data processing

Outputs:

1. A BAM file containing paired singletons error corrected by its complementary SSCS - "sscs.correction.bam"
2. A BAM file containing paired singletons error corrected by its complementary singleton - "singleton.correction.bam"
3. A BAM file containing the remaining singletons that cannot be corrected as its missing a complementary strand - "uncorrected.bam"
4. A text file containing summary statistics (Total singletons, Singleton Correction by SSCS, % Singleton Correction by SSCS, Singleton Correction by Singletons, % Singleton Correction by Singletons, Uncorrected Singletons) - "stats.txt" (Stats pended to same stats file as SSCS)

Concepts:

- Read family: reads that share the same molecular barcode, chr, and start coordinates for Read1 and Read2
- Singleton: single read with no PCR duplicates (family size = 1)

5.4 Duplex consensus sequences (DCS)

`DCS_maker.py`

Function: To generate duplex/double-strand consensus sequences for molecule based error suppression.

- Consensus sequence from bases with Phred quality >= 30
- Consensus quality score from addition of quality scores (i.e. product of error probabilities)

(Written for Python 3.5.1)

Usage: Python3 DCS_maker.py [–infile INFILE] [–outfile OUTFILE] [–bedfile BEDFILE]

Arguments:

–infile INFILE	input BAM file
–outfile OUTFILE	output BAM file
–bedfile BEDFILE	Bedfile containing coordinates to subdivide the BAM file (Recommendation: cytoband.txt)

Inputs:

1. A position-sorted BAM file containing paired-end reads with SSCS consensus identifier in the header/query name
2. A BED file containing coordinates subdividing the entire ref genome for more manageable data processing

Outputs:

1. A BAM file containing paired double stranded consensus sequences - "dcs.bam"
2. A SSCS singleton BAM file containing SSCSs without reads from the complementary strand - "sscs.singleton.bam"
3. A text file containing summary statistics (Total SSCS reads, Unmmaped SSCS reads, Secondary/Supplementary SSCS reads, DCS reads, and SSCS singletons) - "stats.txt" (Stats pended to same stats file as SSCS)

Concepts:

- Read family: reads that share the same molecular barcode, chr, and start coordinates for Read1 and Read2
- SSCS Singleton: a SSCS read without its complementary read