The TagDust2 Manual

by Timo Lassmann

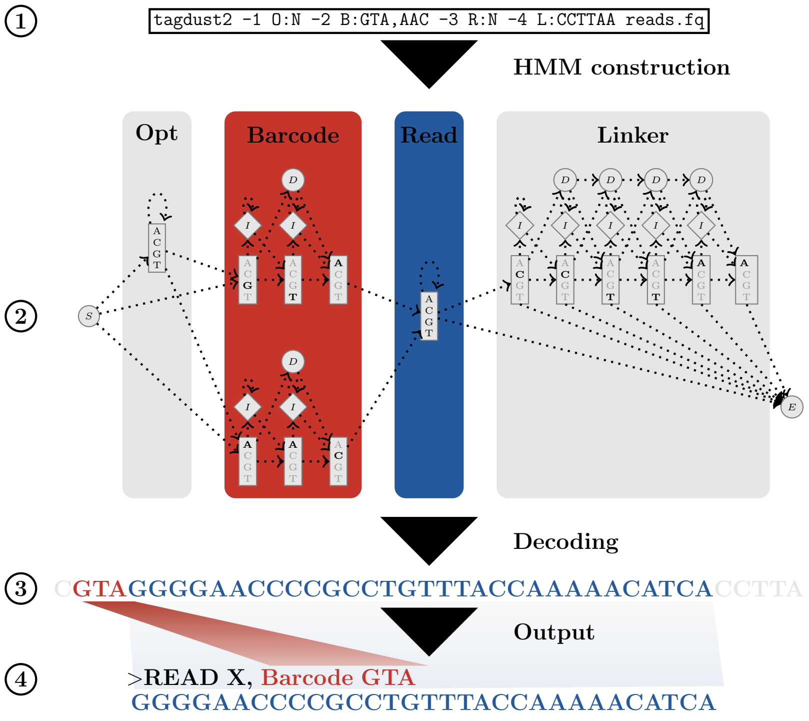
January 6, 2019

Contents

1 Introduction

Raw sequences produced by next generation sequencing (NGS) machines can contain adapter, linker, barcode and fingerprint sequences. TagDust2 is a program to extract and correctly label the sequences to be mapped in downstream pipelines.

TagDust allows users to specify the expected architecture of a read and converts it into a hidden Markov model. The latter can assign sequences to a particular barcode (or index) even in the presence of sequencing errors. Sequences not matching the architecture (primer dimers, contaminants etc.) are automatically discarded (see Figure 1.1).



```
tagdust2 -1 O:N -2 B:GTA,AAC -3 R:N -4 L:CCTTAA reads.fq
```

CGTAGGGGAACCCCGCCTGTTTACCAAAAACATCACCTTA

```
>READ X, Barcode GTA
GGGGAACCCCGCCTGTTTACCAAAAACATCA
```

Figure 1.1: Workflow of TagDust. 1) A user specifies the expected read architecture as a sequence of blocks. In this example there are four blocks. 2) A HMM is constructed by stitching together a set of pre-defined blocks in the order given by the `-1` . . . command line options. For example `-2 B:GTA,AAC` is translated into the second red part of the HMM and models the presence of two mutually exclusive barcode sequences. 3) Each read is scanned with the HMM and each nucleotide is labelled as belonging to a single building block. In the example shown the three letter barcode GTA is recognised in the read. 4) Based on the labelling of the sequence, a barcode is assigned to each read and remaining sequences are trimmed.

2 Installation

Unpack the tarball:

```
bash-3.1$ tar -zxvf tagdust-2.04.tar.gz

bash-3.1$ cd tagdust

bash-3.1$ ./configure

bash-3.1$ make

bash-3.1$ make check
```

At this point the TagDust executable appears in the src/ directory. You can copy it to any directory in your path. To install it system wide type:

```
bash-3.1$ make install
```

3 Quick Usage

3.1 Single-end data

Simply:

```
bash-3.1$ tagdust -arch architectures.txt in.fq -o output
```

The file “architectures.txt” needs to contain at least one valid tagdust architecture. For example:

```
bash-3.1$ more architectures.txt
./tagdust -1 B:AAA,TTT -2 R:N
```

In the case above TagDust will try to split the input reads into those containing either AAA or TTT barcodes at the start of the raw reads.

3.2 Paired-end data

Simply:

```
bash-3.1$ tagdust -arch architectures.txt read1.fq read2.fq -o output
```

4 Usage

4.1 Command Line

TagDust requires an input file containing sequences and a user defined HMM architecture used to extract the reads. The architecture is composed of a selection of pre-defined building blocks representing indices, barcodes, spacers and other sequences one might encounter in the raw output of a sequenced sample. The blocks are supplied to TagDust via command line arguments '`-1,-2, ...`' defining the first, second, . . . block. See the legend of figure 1.1 for more explanation.
Alternatively the architecture(s) can be supplied in a separate input file using the -arch option (see section 4.5.4).

```
bash-3.1$ tagdust <options> <read1.fq>  [read2.fq] -o <output prefix>
```

Option	Type	Description
-arch	STR	path and name of file containing tag dust architectures.
-minlen	INT	minimal accepted read length [16].
-ref	STR	reference fasta file to be compared against[].
-fe	INT	number of errors allowed when comparing to reference[2].
-dust	INT	remove low complexity sequences[].
-o	STR	output prefix.
-t	INT	number of threads [8].
-1	STR	type of the first HMM building block.
-2	STR	type of the second HMM building block.
-. . .	STR	type of the . . . HMM building block.

If two fast files are given TagDust assumes the input is a paired end library.

Here are some examples of HMM building blocks:

Example	Type	Description
R:N	Read	a block containing a read to be extracted.
S:AATT	Spacer	a block containing the sequence AATT.
B:ACT,GGG	Barcode/Index	a three nucleotide long block containing either ACT or GGG
F:NNNNNN	Fingerprint / UMI	a six nucleotide long **U**nique **M**olecular **I**dentifier.
O:T	Optional	a block modeling the optimal inclusion of a single or multiple Ts.

Also have a look at the more detailed description further down and the usage examples.

4.2 Output

TagDust will split the input file(s) into several output files. The number and names of output files depend on the type of library:

	No barcodes	Barcodes
Single End	<p>.fq <p>_un.fq	<p>_BC_GTA.fq <p>_BC_AAA.fq <p>_BC_....fq <p>_un.fq
Paired End	<p>_READ1.fq <p>_READ2.fq <p>_un_READ1.fq <p>_un_READ2.fq	<p>_BC_GTA_READ1.fq <p>_BC_GTA_READ2.fq <p>_BC_AAA_READ1.fq <p>_BC_AAA_READ2.fq <p>_BC_..._READ1.fq <p>_BC_..._READ2.fq <p>_un_READ1.fq <p>_un_READ2.fq

<p> is the prefix specified by the -o option. “**BC**” stands for barcode and “**un**” for un-extracted reads. All auxiliary sequences such as spacers, barcodes are removed from the sequences in the output files.

In addition TagDust summarizes key information about the run in a log file named <p>_logfile.txt. For example:

```
[2014-05-15 16:18:05]   Tagdust 2.10, Copyright (C) 2013 Timo Lassmann
[2014-05-15 16:18:05]   cmd: tagdust-2.10/src/tagdust -arch arch.txt -t 80
-o all_demultiplexed all_raw.fq

[2014-05-15 16:18:05]   Start Run
--------------------------------------------------
[2014-05-15 16:18:05]   Running in single end mode.
[2014-05-15 16:18:05]   Searching for best architecture in file 'arch.txt'
[2014-05-15 16:18:07]   Using: -1 O:N -2 B:TTTAGG,ATTCCA,GCTCAA,CATCCC  -3 S:GGG -4 R:N
[2014-05-15 16:18:07]   Confidence: 1.00
[2014-05-15 16:18:39]   Long sequence found. Need to realloc model...
[2014-05-15 16:21:11]   Selected Threshold:: 1.636996
[2014-05-15 22:24:06]   Done.

[2014-05-15 22:24:07]   all_raw.fq      Input file name.
[2014-05-15 22:24:07]   110622138       total input reads
[2014-05-15 22:24:07]   1.64    selected threshold
[2014-05-15 22:24:07]   100699341       successfully extracted
[2014-05-15 22:24:07]   91.0%   extracted
[2014-05-15 22:24:07]   2529699 problems with architecture
[2014-05-15 22:24:07]   42906   barcode / UMI not found
[2014-05-15 22:24:07]   0       too short
[2014-05-15 22:24:07]   7350192 low complexity
[2014-05-15 22:24:07]   0       match artifacts
```

4.3 Notes

4.3.1 Removing Known Contaminants (e.g. ribosomal RNA sequences).

Known contaminants or unwanted reads can be filtered from the output by pointing TagDust to a reference fasta to be matched against (`-ref <filename>`). The maximum number of errors allowed for a match to reference is set with the `-fe` option. **In the current implementation only the first 32 nucleotides off the extracted read are compared against the reference sequences!**

4.3.2 Removing Low Complexity Sequences

Tagdust implements a simplified version of the DUST module (R. Tatusov and D.J. Lipman, unpublished data) to filter out low complexity reads. While this is not strictly necessary for NGS reads I implemented it to remove poly-A and other "very" low complexity sequences. The algorithm is only applied to the first 64 nucleotides of the reads. The `-dust` option both enables and sets the threshold for the low complexity filtering. I recommend using a relaxed threshold of 100 and leave the rest to the downstream mapping algorithm.

4.3.3 Efficiency

As a default, TagDust attempts to match the entire sequence to the HMM (global). The options `-start` and `-end` restricts the matching to region specified. TagDust assumes that all residues upstream of 'start' do not belong to the read while all residues following 'end' are read residues.

4.4 Formats

4.4.1 Input file formats

TagDust can work with fasta, fastq and SAM/BAM files. Gzipped files are also supported. The format is automatically recognized by the file suffix.

4.5 Examples: putting it all together

4.5.1 A library containing both barcodes and a 8 nucleotide fingerprint.

In this example we expect the sequenced reads to contain a 6 nucleotide barcode sequence (out of a selection of 8) followed by a 8 nucleotide fingerprint, followed by the spacer sequence 'TATA' followed by the read.

Here is the corresponding TagDust command:

```
bash-3.1$ tagdust test.fq.gz   -1 O:N -2 B:ACAGAT,ATCGTG,CACGAT,CACTGA,CTGACG,GAGTGA,GTATAC,TCGAGC
-3 F:NNNNNNNN -4 S:TATA -5 R:N  -o output.fq
```

Note that we added an optional state at the beginning of the read in case extra nucleotides were added before the barcode.

To remove ribosomal reads simply add the -ref and -fe options:

```
bash-3.1$ tagdust test.fq.gz   -1 O:N -2 B:ACAGAT,ATCGTG,CACGAT,CACTGA,CTGACG,GAGTGA,GTATAC,TCGAGC
-3 F:NNNNNNNN -4 S:TATA -5 R:N  -o output.fq -ref U13369.1.fa -fe 2
```

4.5.2 A RNA-seq library with a 'spaced' fingerprint.

Here I downloaded a dataset from [2]. Reads start after the following construct:

```
(NNN)dU(NNNN)dU(NNN)[GACTT]rGrGrGrG
```

dU and rG represent deoxyuridine and guanine ribonucleotide, respectively. N's in brackets are the fingerprint nucleotides while GACTT is the barcode used. As you can see the ten nucleotide fingerprint is split into 3 parts separated by T's.

Here is the corresponding TagDust command:

```
bash-3.1$ tagdust ERR048990.fastq  -1 F:NNN -2 S:T -3 F:NNNN -4 S:T -5 F:NNN -6 B:GACTT
-7 S:GGGG -8 R:N -start 1 -end 25  -o output.fq
```

Note on the last line I restricted the search to the first 25 nucleotides for efficiency.

4.5.3 Extracting multiple reads from a single input.

Here is a simulated case where we have reads at the 5' and 3' end of the input read separated by `GGGGGGGGAAAAAAAA`:

```
@SRR1043305.1 RunXXXXX
ATTGCTGTCTCTTATACACATCGGGGGGGGAAAAAAAATGACGCTGCCCAGATCGTATGCCGTCTTC
+
__bccceefaeeefhfhhhhfhdddhhhaghhhdgffgfYffghfd_cffhffffffffffffffff
@SRR1043305.2 RunXXXXX
GATCGGGGGAACTTTCAGGATCGGGGGGGGAAAAAAAATGTCTCTTATACACATCTGACGCTGCCCA
+
bb_eeeeegfgegiiiiiihhhieffhhfhiihhhffhhifiifiiggbedffffffffffffffff
```

To extract the reads:

```
bash-3.1$ tagdust test.fq -1 R:N -2 S:GGGGGGGGAAAAAAAA -3 R:N -o out
```

The original reads will be split among two output files:

out_READ1.fq	out_READ2.fq
@SRR1043305.1 RunXXXXX	@SRR1043305.1 RunXXXXX
ATTGCTGTCTCTTATACACATC	TGACGCTGCCCAGATCGTATGCCGTCTTC
+	+
__bccceefaeeefhfhhhhfh	fYffghfd_cffhffffffffffffffff
@SRR1043305.2 RunXXXXX	@SRR1043305.2 RunXXXXX
GATCGGGGGAACTTTCAGGATC	TGTCTCTTATACACATCTGACGCTGCCCA
+	+
bb_eeeeegfgegiiiiiihhh	hifiifiiggbedffffffffffffffff

4.5.4 Using the -arch option

When TagDust is given an architecture file containing multiple architectures it will scan a proportion of the input data with all architectures and automatically select the best matching one. For example, the file below contains three architectures:

```
tagdust -1 B:TGCT,AAAA,AACC,AAGG,AATT,ACAC,ACCA,ACGT -2 R:N in.fq -o out.fq
tagdust -1 P:agggaggacgatgcgg -2 B:TGCT,AAAA,AACC,AAGG,AATT,ACAC,ACCA,ACGT -3 R:N
-4 P:gtgtcagtcacttccagcgg in.fq -o out.fq
tagdust -1 P:agggaggacgatgcgg -2 B:TGCT,AAAA,AACC,AAGG,AATT,ACAC,ACCA,ACGT,ACTG,AGAG,AGGA
,AGTC,ATAT,ATCG,ATGC,ATTA,CAAC,CACA,CAGT,CATG,CCAA,CCCC,CCGG,CCTT -3 R:N
-4 P:gtgtcagtcacttccagcgg in.fq -o out.fq
```

Tagdust will scan the file looking for the keyword “tagdust” and assume the remaining words in the line are parameters.
The first two architectures contain eight barcodes / indices but the second one additionally assumes the presence of 5’ and 3’ linkers. The third architecture assumes the presence of 24 barcodes and linkers. When running:

```
bash-3.1$ tagdust -arch architectures.txt in.fq -o output
```

TagDust will automatically select the correct architecture and proceed as normal.
When the input is paired-end data, TagDust will scan both input files with all architectures and attempt to find a match for both reads. **If a match is found for only one file, TagDust will assume the other file only contains valid reads.**

4.5.5 Integration with the Illumina CASAVA pipeline

The CASAVA pipeline converts raw bcl files into de-multiplexed fastq files in a single step. To use TagDust2 with Illumina data the default de-multiplexing has to switched off by adding the option:

```
--use-bases-mask Y*,Y6 ,Y*
```

to the configureBclToFastq.pl command and by modifying the sample-sheet. Increase 6 in the example above to the actual length of the barcodes in case they are longer. For more detail have a look at the guide by Brant Faircloth[1] and have a look at the CASAVA documentation.
Now run TagDust on all output files using the following *architecture file* (see section 4.5.4):

```
tagdust -1 B:ACAGTG,ACTTGA,TTAGGC
tagdust -1 R:N
```

The first line tells TagDust2 to look for reads composed entirely of one of the three index sequences. This line has to be changed to reflect the actual indices used in a particular experiment. It is also possible to add additional lines and let TagDust2 figure out which combination of indices was used. The second architecture is used for both the other actual reads. Here is the actual command:

```
bash-3.1$ tagdust -arch architectures.txt not_demultiplexed_NoIndex_L001_R1_001.fastq.gz
not_demultiplexed_NoIndex_L001_R2_001.fastq.gz not_demultiplexed_NoIndex_L001_R3_001.fastq.gz
-o demultiplexed_
```

[1] https://gist.github.com/brantfaircloth/3125885

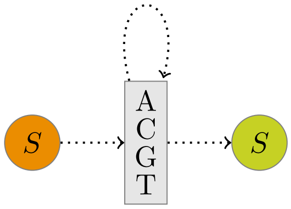
5 HMM Building Blocks

TagDust comes with a set of pre-defined HMM building blocks. Each includes a silent state at the beginning and end used to link blocks together. Each block is specified by a unique letter following by a colon and some information about the sequence.

5.1 Read

Segment modeling the read.

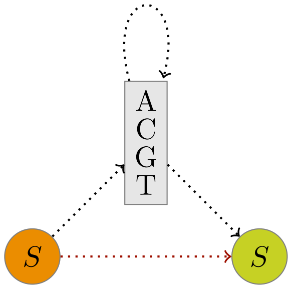
Code: -[1-9] R:N



5.2 Optional

Segment modeling an optional single or short stretch of nucleotides.
Example code: O:N

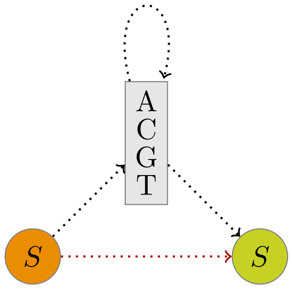
Code: -[1-9] O:N



5.3 G addition

Segment modeling the occasional addition of guanines to the reads. (89.3% chance of a single G , 19.5 % chance of 2 Gs..).
Example code: O:N

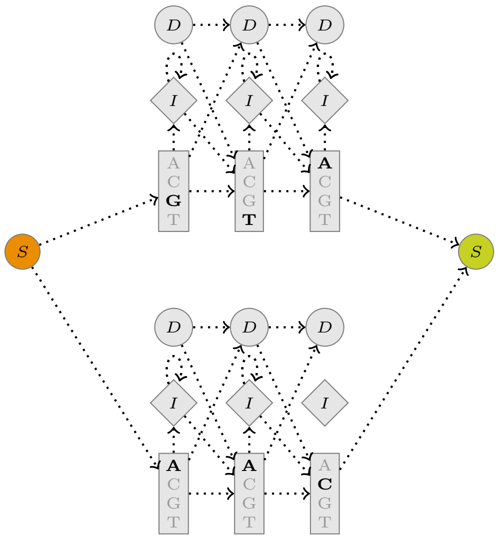
Code: -[1-9] G:G



5.4 Barcode or Index

Segment modeling a set of barcode sequences. For each sequence a separate HMM is created. The barcode sequences must be given as a comma separated list. A null model of the same length as the barcode is automatically added and initialized to the background nucleotide frequencies.

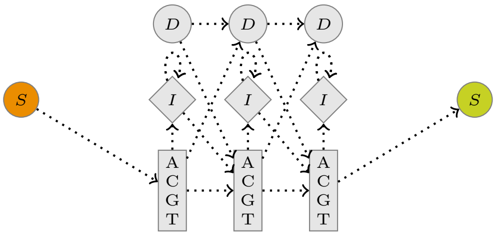
Code: -[1-9] B:GTA,AAC



5.5 Fingerprint or Unique Molecular Identifier - UMI

Segment modeling a fingerprint (or unique molecular identifiers). Insertions and deletions are by default not allowed within a fingerprint segment.

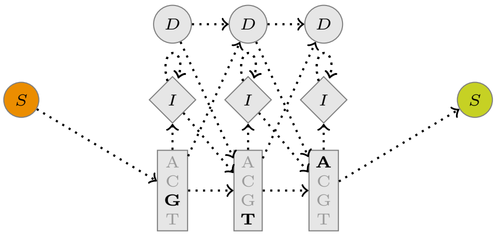
Code: -[1-9] F:NNN



5.6 Spacer

Segment modeling a pre-defined sequence.

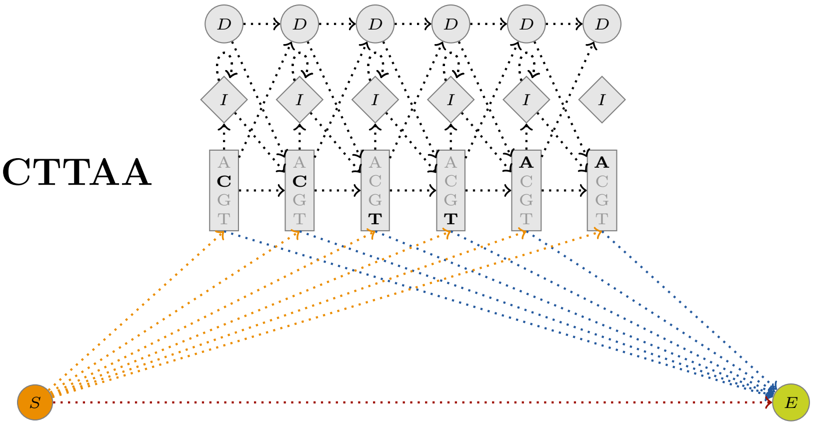
Code: -[1-9] S:GTA



5.7 Partial

This segment is used to model sequences that may only be partially present at the 5‘ or 3‘ end of the read. The transition probabilities (orange and blue) are set automatically based on the length distribution of **exactly** matching adapters.

Code: -[1-9] P:CCTTAA



6 Algorithm

6.1 Sequence scoring

In short read mapping the mapping quality Q reflects the confidence we have in one particular mapping location over all others [3]. Analogously, TagDust2 compares the probability of each read matching to the user specified HMM to the total summed probability including a random model:

$$P = 1 - \frac{P(x|M) * V}{P(x|M) + P(x|R)} \tag{6.1}$$

where $P(x|M)$ is the total summed probability of a read matching the model derived by the forward algorithm and $P(x|R)$ is the probability of the read give a random zero order Markov model. V represents the fraction of $P(x|M)$ corresponding to the most likely barcode sequence. It is derived by using the backward and forward algorithm at each mutual exclusive transition:

$$V = \max_j \left(\frac{f_s(i) a_{s,m_j} e_{m_j}(x+1) b_{m_j}(i+1)}{\sum_\pi P(x, \pi|M)} \right) \tag{6.2}$$

where a_{s,m_j} is the transition from a silent to the first match state m_j of a HMM modeling a barcode.

6.2 Threshold selection

In versions prior to 2.11 users could specify a threshold with the -q option. Unfortunately, when using simple read architectures (e.g. 48, 3bp barcodes followed by a read) often no reads obtained a reasonable P (equation 6.1).
TagDust now simulated reads based on the model and random model and selects a threshold automatically giving the best specificity plus sensitivity.

6.3 Implementation

Internally, TagDust uses a full profile HMM for each building block. We simply set some transition probabilities to zero to emulate different models. For example the read building block 'R' is implemented as a profile HMM with one column and transitions directly to and from the insertion state. This is fairly inefficient as the dynamic programming code still needs to evaluate all possible transitions, even those we know in advance have a probability of zero. In future I might do something about this.

6.4 Optimal accuracy decoding

To obtain the most probably labeling of the sequence, we employ the optimal accuracy decoding algorithm as described in [1]. To apply this algorithm to our problem define the label probability of a nucleotide by the summed posterior label probabilities of states belonging to a particular HMM building block. A secondary dynamic programming algorithm is used to determine the path with the maximal label probability, constrained by the global HMM architecture. The label probabilities are essentially used as a substitution matrix while the architecture is enforced by the equivalent of gap penalties.

If fingerprints are present TagDust checks at this stage if the length after decoding matches the users input. If not the read is discarded.

Bibliography

[1] Lukas Käll, Anders Krogh, and Erik L L Sonnhammer. An HMM posterior decoder for sequence feature prediction that includes homology information. *Bioinformatics (Oxford, England)*, 2005.

[2] Teemu Kivioja, Anna Vähärautio, Kasper Karlsson, Martin Bonke, Martin Enge, Sten Linnarsson, and Jussi Taipale. Counting absolute numbers of molecules using unique molecular identifiers. *Nature methods*, 9(1):72–74, January 2012.

[3] Heng Li, Jue Ruan, and Richard Durbin. Mapping short dna sequencing reads and calling variants using mapping quality scores. *Genome research*, 18(11):1851–1858, 2008.