TEDNA Manual

Matthias Zytnicki

December 19, 2018

1 Introduction

Tedna is a simple transposable elements assembler. It reads a two FASTQ files, representing paired-end DNA-Seq data, and produces a FASTA file with the transposable elements it has assembled.

2 Tedna at a glance

- Compile with **make**
- Run Tedna: **./tedna -k 61 -1 *left*.fastq -2 *right*.fastq -i *insert_size* -o tes.fasta**

3 Install

3.1 Requirements

You will need C++11 to compile Tedna. Nothing else is required. Type **make** to compile.

C++11 is a new standard of C++. Older compilers do not support this version. For GCC, compilers older than version 4.6 will not work. You can also compile with Clang. Version 3.0 should also work.

3.2 Hash implementation choice

Tedna makes heavy use of a hash table. Three of them are offered: sparsehash-sparse, default C++11 hash, and sparsehash-dense. The first and the last are provided by the sparsehash package, and the second one is provided by C++11. The first implementation is the slowest one, but requires least memory, whereas the last implementation is fast but requires lots of memory. The second one is somewhere between the two. Choosing which implementation is up to you, but you have to choose at compile time. Type **make HASH=SLOW**, **make HASH=MID**, or **make HASH=FAST** to choose your implementation. By default, the slowest implementation is chosen.

3.3 *k*-mer size choice

As you will see by reading the rest of the manual, Tedna relies on the choice of a good *k*-mer size. The maximum size is determined at compilation time. For instance, if you compile with the option **make k=91**, you will be able to use *k*-mers up to 91 nucleotides. However, note that compiling with large *k*-mer sizes leads to a program which will use substantially more RAM.

By default, the size is set to **61**.

4 Options

4.1 General behavior

This manual describes each step of Tedna's execution, and describes in detail all the parameters it uses. There is no hard-coded parameter in Tedna. The most useful parameters have a long version (**--help**) and a short version (**-h**). Compulsory options are denoted with a start (⋆). For instance, a brief description of the options can be obtained with **./tedna -h** or **./tedna --help**.

Tedna reads one or two FASTQ files.

-1, --file1⋆ first FASTQ file.

If you use paired-end reads, the second file should be the second read of the fragment, and the insert size between the paired-ends reads should also be provided:

-2, --file2 second FASTQ file (if you use paired-end reads),

-i, --insert the insert size.

Tedna then builds a de Bruijn graph, a data structure which has been often used in assemblers. De Bruijn graphs collect the so-called *k*-mers: the set of sub-words of size *k* you can find in the reads.

-k, --kmer⋆ the *k*-mer size.

It is specially crucial and difficult to find a good *k*-mers size. Trial and error usually is a good option. If you are in a hurry, you could consider trying 61.

Tedna finally creates a new FASTA file, with the transposable elements it has found.

-o, --output⋆ the FASTA output file.

4.2 First details

By default, Tedna tries to infer the coverage of the genome (also known as the depth of the sequencing), i.e. the number of times a particular nucleotide of the genome has been sequenced. Then, Tedna focusses its attention to the repeated sequences, i.e. the sequences that appear twice as often as the genome sequence. You can choose to assemble sequences that appear thrice as often, etc.

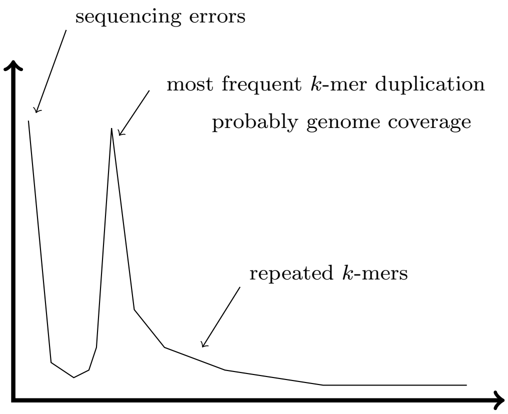

Figure 1: An ideal k-mer frequency distribution. A point (x, y) means that y different k-mers have been found x times in the reads. k-mers on the very left of the distribution are supposed to be sequencing errors: every k-mer is found only once or twice. The maximum of the distribution is supposed to be the genome coverage. A genome with more than 50% of repeated sequences is not expected to have that shape, and thus Tedna will not be able to find the genome coverage. A sequencing with many errors may also confuse Tedna.

--repeatFrequency the relative frequency of the sequence you seek to assemble.

However, Tedna uses a very crude way to estimate the coverage of the genome: it takes the maximum of the k-mer frequency distribution (see Fig. 1). As a consequence, Tedna might also fail to find the genome coverage. In this case, you should provide it with the estimate transposable element proportion.

-t, --threshold repeated fraction of the genome (expressed as percentage).

You can also control the size of the transposable elements you have assembled, so as to remove uselessly small elements, or suspiciously long ones.

-m, --min-te-size minimum transposable element size

-M, --max-te-size maximum transposable element size

Finally, if you have several processor and you are ready to use them, you can specify it to Tedna. Most of the pipe-line may use several processors.

-p, --processors number of processors

4.3 Other details

4.3.1 Reads parsing

If you want to accelerate the assembly or reduce the RAM consumption, you can decide to read only the N first reads of you file. The results should be given faster, at the price of a degraded result quality.

--max-reads reads read.

If you use several processors, each one will read several blocks of some fixed size during the FASTQ read step. Actually, any reasonable size could do, but you can change it.

--bytes-per-thread size of the blocks read by each processor.

4.3.2 Assembly

After having built the de Bruijn graphs, Tedna decomposes it into connex components. Every connex component should be a transposable element. However, very small components are most likely short repeated sequences.

--small-graph minimum size for a component. Smaller component are discarded.

On the contrary, you can end up with very large components. If they represent sequences longer than 20k, you are most likely assemblying the whole chromosome. You should then decrease the value of the parameter **--threshold**.

--big-graph maximum size for a component. Tedna stops when it tries to assemble such a big component.

The decomposition in connex components is performed on the fly, and each time a new component is discovered, it is analyzed and assembled. When Tedna sees many small components in a row, it supposes that no valid component remains.

--small-graph-count stop the decomposition have having found N small components in a row. Use **0** if to deactivate this feature.

Some neighbor nodes may have different frequencies. It is usually a hint that the nodes are actually not neighbors, and that the link between them is an artifact.

--frequency-dif maximum frequency differency beetween two neighbors.

Sequencing errors, as well are rare transposable elements polymorphisms, tend to artificially increase the size of each component. If it were to assemble the uncurated components, Tedna would spend much too much time. As a consequence, Tedna first erode the tips of the components. The eroded tips are leaves of the components that bifurcate from long paths, and are thus likely to be unwanted sequences.

--erosion maximum size of the tips (in nucleotides) to remove.

Second, Tedna also pinch bubbles. Bubbles are formed in the graph in case of polymorphism. When a bubble is detected, one of its paths is cut open.

--bubble-size maximum size of a bubble.

After having analyzed a component, Tedna produces a set of possible transposable elements.

4.3.3 LTR detection

LTR elements have long, near identical, sequence ends. As a consequence, a component representing an LTR is likely to form a loop. Tedna implements special algorithms to detect these loops.

--min-ltr minimum size of an LTR (only the long terminal repeat).

--max-ltr maximum size of an LTR.

4.3.4 Duplication removal

When two transposable elements have sufficiently diverged, two different variants may end up in to different components. Tedna removes these duplicates:

--duplicate-id percentage of divergence to declare 2 elements different.

4.3.5 Merging

When some part of a transposable elements has been little sequenced, or when a region is highly polymorphic, one transposable element may end up in two different components. Tedna then tries to merge those parts of transposable elements such that their sequence ends have a high similarity.

--min-overlap minimum size of the sequence end to be merged,

--max-overlap maximum size of the sequence end to be merged.

Tedna uses a variation of the simple Needleman–Wunsch algorithm to detect similarity on the sequence ends.

--indel-pen penalty for an insertion/deletion in the alignment,

--mismatch-pen penalty for a mismatch in the alignment,

--max-pen maximum penalty to declare a match.

Tedna tends to favor longer similarities than short ones: it will give higer penalty score for a perfect match of size 10 than to a nearly perfect match of size 100.

--size-pen penalty for a short alignment.

Tedna finally merge the ends if they also have a high similarity.

--min-id minimum identity to declare a match.

Before filling the big Needleman–Wunsch matrix, Tedna first scan the k-mers of two sequences to compares. If the intersection of the k-mers sets is empty, the sequences are not merged before any computation. To be efficient, the size of these k-mers should be small.

--short-kmer size of these k-mers.

4.3.6 Scaffolding

Tedna finally uses the paired-end information to scaffold the transposable elements parts. It reads the k-mers contains in each transposable element part, then reads every paired-end. Every k-mer pair, found both in the paired-ends, and in two transposable elements parts, is counted as one evidence.

--min-scaffold minimum number of evidences to form a scaffold,

--max-scaffold maximum number of evidences to form a scaffold.

In some cases, the scaffolder collects evidences for joining too many transposable element parts: the scaffolder may virtually join any part to any part. This usually is a hint that the minimum number of evidences used to scaffold pairs is too low. An other parameter raises the minimum number of evidences, so that a transposable part may not be potentially joined with more than N other neighbors.

--scaffold-max-nb maximum number of neighbors per transposable element part.

4.3.7 Probably useless details

--bytes-per-thread Number of bytes each thread reads in the FASTQ files.

--max-reads Maximum number of reads used for the assembly.

--check Provide a sequence, and Tedna will track your sequence during the assembly process. Mostly for debugging purposes, but it may be useful if are dissatisfied with the way Tedna assembled your transposable element.

5 Output

Tedna outputs a multiple-FASTA file. Each sequence is a putative transposable element. The header is formatted as follows: `>te_n_freq:m (s)`, where n is the id of the transposable element, m is the average number of k-mers per full-length transposable element, and s is the size of the transposable element.

The sequences may contain stretches of `N`s. These stretches are produced during scaffolding. Suppose that we have a sequence A, followed by m `N`s, and followed by a sequence B. This means that, considering paired-ends information, the sequence A and B should be separated by about m bp, although the content of the sequence between A and B is not known.

6 Troubleshooting

I cannot compile Tedna

- First check that your compiler can process C+11.

- There is a bug in GCC v. 4.8.1 when using threads: `terminate called after throwing an instance of 'std::system_error' what(): Enable multithreading to use std::thread: Operation not permitted`. To circumvent the problem, you can add **-Wl,–no-as-needed** in the flags of the makefile if you experience this problem.

- Last resort, download binary files from the Web site: `http://urgi.versailles.inra.fr/tools/tedna/` (I hope they are compatible with your architecture).

Tedna is too slow

- Try first to reduce the number of reads read with the option **--max-reads**.

- You could also decrease the overall threshold (option **-t**).

- Increasing the size of the *k*-mers (**-k**) usually accelerates the assembly.

- If you have enough RAM, you can try to compile with another hash implementation: compile with **make HASH=FAST**.

- If Tedna takes too much time at some given stage, try to modify the parameters that concerns this stage:
  - during initial assembly: reduce the maximum graph size (**--big-graph**),
  - during sequence ends merge: increase the identity threshold (**--min-id**),
  - during scaffold: increase the minimum number of evidences (**--min-scaffold**).

Tedna needs too much memory

- First check that you chose the right hash implementation (compiled with **make HASH=SLOW**).

- You can also choose a smaller *k*-mer size and recompile Tedna accordingly.

- You can also reduce the number of reads read with the option **--max-reads**.

- Decrease the overall threshold (option **-t**).

Tedna does not find my preferred transposable element Tedna uses lots of heuristics, and one of them may skip your transposable element. If so, you can use the **--check=*my_transposable_element_sequence*** option. Tedna will track your transposable element along the assembly process. Hopefully, it will provide you the reason why your transposable element was not assembled.

Tedna stops after `Cannot determine maximum peak distribution. Please provide an expected repeated genome coverage.` Tedna may not be able to find the genome coverage (see Fig. 1) for two reasons:

- Your genome is TE-rich. The peak at the genome coverage does not stand out. You should then specify the expected genome coverage with the **`-t`** parameter.
- The quality of the sequencing is poor (you can check this with FastQC). You should then clean your reads by trimming the 3' ends or discarding poorly sequenced reads (e.g. with the FASTX-Toolkit). I do not endorse any third-party tool.

7 Contact

In case of question, suggestion for improvement, or any other enquiry, do not hesitate to contact me: matthias.zytnicki@inra.fr.