TARGet

User's Guide

May 17, 2015

1 Introduction

TARGet is a light-weight application for reconstructing non-vertical evolutionary histories from sampled genetic sequences. TARGet makes use of persistent homology [1,2], a tool from topological data analysis, to infer information about the minimal set of reticulate events (recombination, re-assortment, etc) that are needed to explain a sequence alignment, under the assumption of no convergent evolution. It exploits a sequence partitioning strategy similar to that of ref. [3], but the topological nature of TARGet permits inferring phylogenetic information about reticulate events, like the participating gametes or the associated genetic scales. Specifically, TARGet computes the extended first-homology barcode of the sequence alignment and attempts to reconstruct the topological ancestral recombination graph (tARG), which is closely related (in most cases actually identical) to minimal ARGs [4].

The advantage of working with barcodes and tARGs is that they can be computed in polynomial time, without having to perform exhaustive explorations of the ARG space. Hence, as opposed to standard methods of ancestral recombination graph (ARG) reconstruction, TARGet is a fast multi-threaded algorithm that can be applied to large datasets, consisting of hundreds of genetic sequences. It is therefore a great tool for exploring reticulate evolution not only in virus and bacteria, but also in eukaryotes.

TARGet is distributed under the GNU General Public License (GPL v3). It is fully written in Python 2.7, but heavily relies on Dionysus C++ library for persistent homology computations. It runs in Windows, Mac OS and Unix systems, provided all dependences are installed. If you use TARGet for your research, please include [4] in your references.

Installation

To run TARGet, the following free software is required in the system:

- Python 2.7. (Python 3 is currently not supported.)
- CMake.
- Boost C++ libraries, including Boost.Python.
- NetworkX package for Python.

For optimal visualization results it is also recommended to have Graphviz tools and PyGraphviz installed, although they are not strictly required.

Once the above dependences are installed in the system, you can unpack and build TARGet by typing the following commands in a terminal:

```
tar -xvzf TARGet.tar.gz
cd TARGet
mkdir build
cd build
cmake ..
make
```

The Python executable is in directory `TARGet`. You can check that TARGet has been correctly installed by running it on the test or on the example FASTA files that come with the distribution:

```
./TARGet test.fa
```

or

```
python TARGet test.fa
```

Troubleshooting

These are some potential problems that can happen during the installation procedure, and their solution:

- On Mac OSX, CMake often has issues finding the correct python path. If that is the case, uncomment the following three lines in file CMakeLists.txt, replacing the directories for the appropriate ones in your system,

  ```
  set (PYTHON_EXECUTABLE /usr/local/bin/python2.6)
  set (PYTHON_INCLUDE_DIR /usr/local/include/python2.6)
  set (PYTHON_PATH /usr/local/lib/libpython2.6.dylib)
  ```

  and run `cmake` and `make` again.

- Some compiler versions may raise the following error:

  ```
  error:  unrecognized command line option "-ftemplate-depth=256"
  ```

  If that is the case, replace the following line in CMakeList.txt,

  ```
  add_definitions         (-ftemplate-depth=256)
  ```

  by,

  ```
  add_definitions         (-ftemplate-depth-256)
  ```

  and run cmake and make again.

- It is important to run TARGet using the same version of Python that was used to compile TARGet and Boost libraries.

2 Running TARGet

Input file

The standard TARGet input is a FASTA file containing a DNA sequence alignment. For better performance, it is recommended to remove duplicated sequences from the input file before running TARGet. The current version cannot deal properly with large gaps (more than 2 or 3 bases long), so it is also advised to remove those loci from the alignment. A suitable gap treatment will be implemented in next versions of TARGet.

Alternatively, the input file can contain only segregating sites (in bi-allelic 0/1 format). For instance:

```
>Sequence ID 1
11110000
>Sequence ID 2
00001100
...
```

Note, however, that all positions in TARGet's output will refer in that case to segregating sites and not to genomic positions.

Command line options

TARGet can be run with default parameters by just typing:

```
./TARGet <input file>
```

or

```
python TARGet <input file>
```

where `<input file>` is the name of the FASTA file containing the alignment. However, there are several command line options that may result useful:

`-c #`

TARGet is a multi-threaded algorithm. By default it will run in a single CPU core, but you can specify a higher number of cores with the command line option `-c #`, where `#` is the number of cores that TARGet can use.

`-o <prefix>`

TARGet produces four output files. By default these are named:

```
out.b1.txt
out.bars.txt
out.targ
out.gexf
```

A different prefix (instead of `out`) for the output files can be specified with the command line option `-o <prefix>`.

`-s #`, `-t #` and `-w #`

As already stated, TARGet makes use of a sequence partitioning strategy similar to that of Myers and Griffiths [3]. In most cases, considering all possible sequence partitions is unnecessary and computationally too expensive. The number of sequence partitions that TARGet computes can be limited by the maximum and minimum numbers of segregating sites within each segment, and by the maximum distance (measured in number of segregating sites) between segment extrema. The default values for these three quantities are 2, 5 and 13, respectively. The user can specify different values by means of the command line options `-s #`, `-t #` and `-w #`, respectively, where `#` indicates the number of segregating sites.

In general, taking large values for `-s #` and `-w #` can increase substantially the running time and memory requirements. We find that a useful strategy is to first run TARGet with default parameters. Then, increase as much as possible the number of segregating sites within segments using `-s #`. This will improve the sensitivity and the genetic scale range. If the value of `-s #` approaches that of `-w #`, and running time and memory requirements are still good, you may want to also increase the maximum distance between segment extrema using `-w #`. Sometimes you may capture large scale recombination loops, even if they could be split in smaller ones. Increasing `-t #` is a good strategy in those situations.

`-e`

It excludes from the computation segregating sites that are compatible with all the other sites in the sequence alignment. This speeds up the computation, but can slightly alter the genetic scales of some of the bars in the extended first-homology barcode.

`-n`

By default, when TARGet finishes computing the extended barcode and tARG of the sequence alignment, it opens a graphical interface that allows exploring the results interactively. The use of the graphical interface is described below. The user may want to change this behavior, so that the graphical interface is not automatically opened. This can be achieved by means of the command line option `-n`. Note that it is always possible to resume previous results by using the command line option `-l`, described below, without having to recompute the tARG.

`-l`

One of the three output files that TARGet produces is a binary file (by default named `out.targ`) containing all the information required to open a previously computed extended barcode and tARG in the graphical interface. You may use the command line option `-l` to open a `.targ` file. In this case the input file is not a FASTA file, but a `.targ` file.

3 Understanding the output

TARGet produces an extended barcode associated to the sequence alignment and attempts to reconstruct the tARG. Each bar in the extended barcode represents a reticulate event in the sample history. Bar positions and lengths are related to the genetic scales of the gametes that took part in the reticulate event, measured in number of mutations. Assuming good sampling density, reticulate events involving genetically very different parental sequences (e.g. recombination of strains that diverged long time ago) are represented by long bars at high genetic scales in the barcode, whereas reticulate events involving closely related parental sequences appear as short bars at low genetic scales.

For the sake of illustration, consider a sample of 7 sequences with the following genotype (these can be found in the file `test.fa`):

```
0 :  1 1 1 1 0 0 0 0
1 :  0 0 0 0 1 1 0 0
2 :  0 0 0 0 1 1 0 1
3 :  1 1 1 1 1 1 1 0
4 :  1 1 1 1 0 0 1 0
5 :  0 0 0 0 0 0 0 0
```

```
6 : 1 1 1 1 1 1 0 0
```

Minimal histories of the sample involve 2 recombination events under the assumption of no convergent evolution. An example of minimal history is shown in figure 1a. The corresponding tARG, depicted in figure 1b, represents the unrooted topology of minimal ARGs [4]. Hence, it contains the two loops present in the minimal ARG of figure 1a, and allows identifying the specific gametes that participate in each recombination event.

The extended barcode of the sample is shown in figure 1c. It has two bars, corresponding to the two recombination events of minimal ARGs. The shortest bar is associated to the recombination between gametes 0 and 3, which differ on a single nucleotide. The longest bar is associated to the recombination between gametes 0 and 1, which differ in 6 nucleotides. The location of recombination break points is also shown in figure 1c.

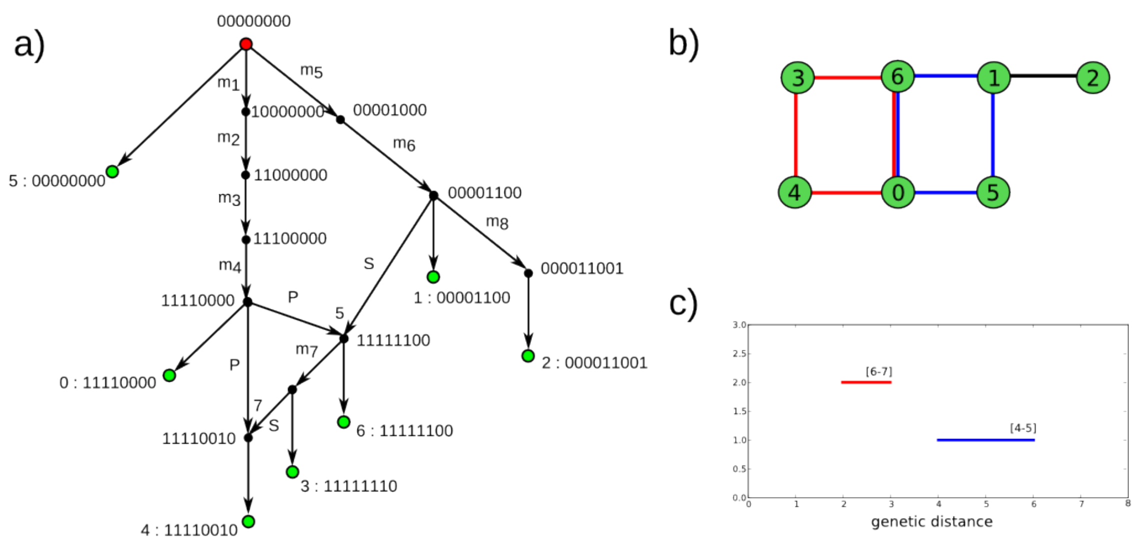


Figure 1: a) Example of minimal ARG describing a minimal evolutionary history for the sample of 7 sequences considered in the text. Mutation events in the i-th position are represented as m_i. Root node is marked in red and leaf nodes are marked in green. The ARG involves two recombination events. b) tARG associated to the sample. c) Extended barcode associated to the sample.

These results can be simply obtained by running

```
./TARGet -s 8 test.fa
```

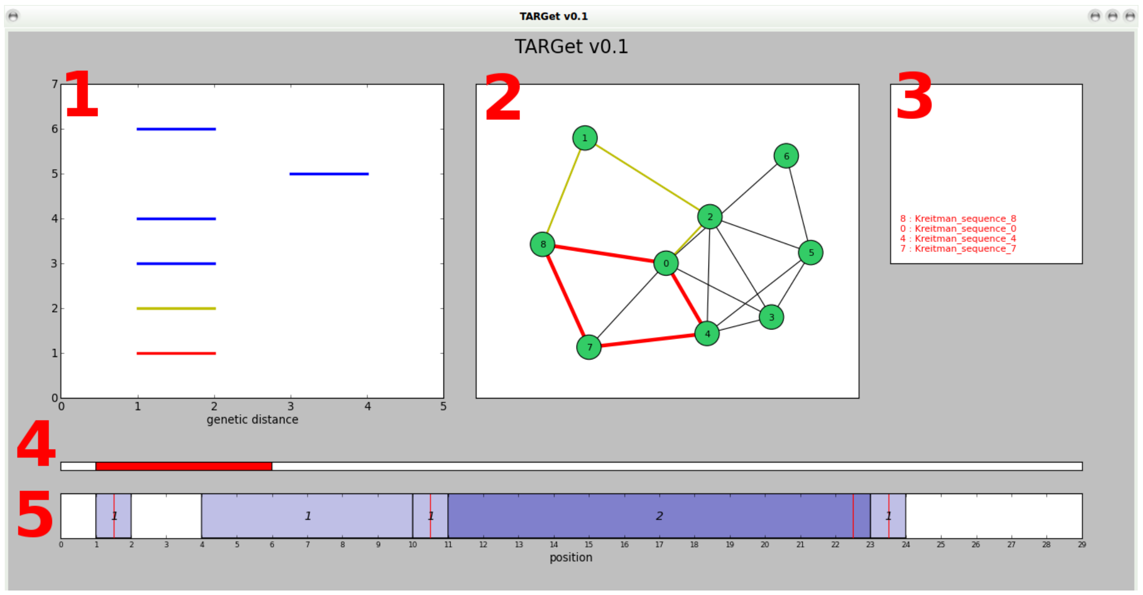
Barcodes are relatively stable against sampling. For instance, in the above example we would obtain the same barcode if we remove sequences 0, 2 and 6 from the sample.

Graphical interface

TARGet has a interactive graphical interface that permits exploring the results. A snapshot of the interface is shown in figure 2. The interface consists of 5 different panels. Panel 1 contains the extended barcode of the sequence alignment, depicting genetic scales of reticulate events, as described above. Panel 2 contains the reconstructed tARG of the sample. Panel 5 represents a set of intervals on the genetic sequence, indicating the number of reticulate events of the tARG (the so-called *first Betti number*) with break points in the interval. In addition, loci at which the Hudson-Kaplan's four gamete test [5] fails across consecutive segregating sites are also indicated in panel 5 (red lines). Positions in that panel refer to genomic positions, except when the input file contains only segregating sites.

When selecting a bar in panel 1, the corresponding loop is marked red in the tARG representation of panel 2, and panel 3 list the identifiers of the genetic sequences that participate in that reticulate event (see figure 2).

Generally, the extended barcode is built out of various sequence blocks [4], following a partitioning algorithm similar to that of [3]. Hence, when selecting a bar in panel 1, other bars belonging to the same building block are also highlighted in yellow, as well as the corresponding cycles in panel 2. In addition, panel 4 indicates the part of the genetic sequence that is associated to that building block.



```
8 : Kreitman_sequence_8
0 : Kreitman_sequence_0
4 : Kreitman_sequence_4
7 : Kreitman_sequence_7
```

Figure 2: Graphical interface of TARGet.

Output files

TARGet produces four output files, by default named `out.bars.txt`, `out.b1.txt`, `out.targ` and `out.gexf`. We have described above the use of the binary file `out.targ`, containing all the information required to resume a graphical session.

The file `out.bars.txt` contains a tab separated table listing all reticulate events in the extended barcode. The format is:

```
birth	death	generators	start	end
```

Each row represents a single reticulate event in the tARG. The first two columns in the table contain the genetic scales of the two bar extrema in the extended barcode. The third column contains a list of the edges that make the corresponding loop in the tARG. Each edge is given by a pair of sequences, where sequences are denoted by integers starting from 0 in the same order as they appear in the input file. The last two columns contain respectively the start and end positions of the genomic segment associated to the building block containing the reticulate event [4].

The file `out.b1.txt` contains a tab separated table with columns:

```
start	end	b1
```

Each row represents an interval of the genetic sequence constraining as much as possible the location of reticulate events break points. The first and second columns of the table contain the start and end positions of the interval, respectively. The third column is the number of reticulate events with break points in that interval. Note that currently the algorithm that TARGet uses to constrain the location of break points is sub-optimal, and in complex situations some of the reticulate events listed in the file `out.bars.txt` may be missing in the file `out.b1.txt`.

All positions listed in the columns `start` and `end` in files `out.bars.txt` and `out.b1.txt`, refer to genomic positions except when the input file contains only segregating sites.

Finally, the file `out.gexf` contains the reconstructed tARG in GEXF (Graph Exchange XML Format), that can be opened by most network analysis programs.

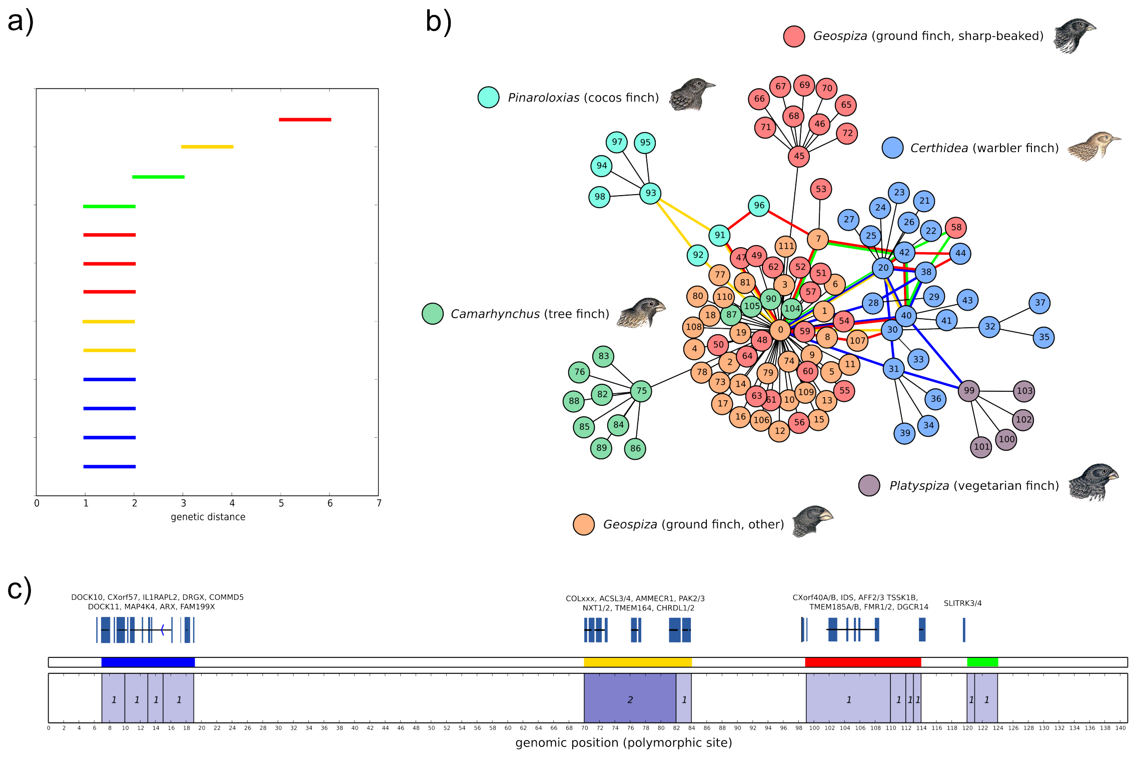


Figure 3: Extended first-homology barcode and reconstructed tARG of a sample of 112 Darwin's finches, obtained with `TARGet` (the source FASTA file, `Darwing_Finches.fa`, is included in the distribution). The extended first-homology barcode is shown in a), based on 140 homozygous SNPs present in a 9 megabase scaffold. In total, 13 recombination/gene flow events are captured in the extended barcode, having different genetic scales. Bars are coloured according to the position of the corresponding recombination breakpoint in the genome, as depicted in The number of recombination events detected at each genomic interval, as well as some of the orthologous genes present at regions where recombination events are detected, are indicated in c). The reconstructed tARG is presented in b). Recombination loops in the reconstructed tARG are outlined using the same code of colors. Leaf nodes that do not participate in any recombination event are also included in the graph, using a nearest neighbour algorithm based on genetic distance. Edge lengths are arbitrary. Figure taken from [4].

References

[1] Edelsbrunner, H., Letscher, D. and Zomorodian, A. (2002), *Topological persistence and simplification*. Discrete and Computational Geometry **69**, pp. 511-533.

[2] Zomorodian, A. and Carlsson, G. (2005), *Computing persistent homology.* Discrete and Computational Geometry **33**, pp. 247-274.

[3] Myers, S.R. and Griffiths, R.C. (2003), *Bounds on the minimum number of recombination events in a sample history.* Genetics **163**, pp. 375-394.

[4] Camara, P.G., Levine A.J. and Rabadan R. (2015), *Inference of ancestral recombination graphs through topological data analysis.* In preparation.

[5] Hudson, R.R. and Kaplan, N.L. (1985), *Statistical Properties of the Number of Recombination Events in the History of a Sample of DNA Sequences.* Genetics **111**, pp. 147-164.