Decombinator: a tool for fast, efficient gene assignment in T cell receptor sequences using a finite state machine

Niclas Thomas [1], James Heather [2], Wilfred Ndifon [3], John Shawe-Taylor [4], Benjamin Chain [2*]

[1]CoMPLEX Department, UCL, Gower Street, London, WC1E 6BT, UK
[2]Division of Infection and Immunity, The Cruciform Building, UCL, Gower Street, London, WC1 6BT, UK
[3]Department of Immunology, Weizmann Institute of Science, Rehovot, 76100 Israel
[4]Department of Computer Science, UCL, Gower Street, London, WC1E 6BT, UK

*Corresponding Author: Tel: 020 31082129, Email: b.chain@ucl.ac.uk

Introduction

Decombinator is a tool for the fast, efficient analysis of T cell receptor (TcR) repertoire samples, designed to be accessible to those with no previous programming experience. It is based on the Aho-Corasick algorithm which uses a finite state automaton (FSA) to quickly assign a specific V and J gene segment. From these assignments, it is then able to determine the number of germline V and J deletions and the string of contiguous nucleotides which lie between the 3' end of the V gene segment and the 5' end of the J gene segment. These 5 variables form the identifier which uniquely categorises each distinct TcR sequence. For more details, please see (Thomas et al.). Decombinator assumes no prior programming experience.

System Requirements

Decombinator requires the following applications: -

- Python v2.6 or v2.7
- BioPython
- NumPy

It has been successfully tested on Windows 7 and Linux platforms, with Python v2.6 and v2.7, but is untested on v3.x.

Getting Started

Install the following components in the order they appear on the list:-

- To install Python, please see `http://www.python.org/getit/`.
- To install NumPy, please see `http://numpy.scipy.org/`.
- To install Biopython, please see `http://biopython.org/wiki/Download`.

It is worthwhile adding Python to your PATH to be able to access it easily via the command prompt (alternatively you may wish to use a Python Shell such as IDLE). To add Python to your system's PATH in Windows: -

- Right-click *My Computer* and click *Properties*
- In the left panel, click *Advanced system settings*
- Click on *Advanced* and then *Environment Variables*
- In the box under *System variables*, click on *Path*, and click *Edit*.
- To the end of the list, add ;$C:\Python27$ (no spaces, and note the ; at the front), or the equivalent for your installation of Python. To find the location of installation, you can usually type *python* into the Windows search box, right-click and select *Properties* on the Python icon that appears at the top of the list.

To check this has been successful, open the command prompt (i.e. type *cmd* into the Windows search box and press *Enter*), type *python* and press *Enter*. If this opens the Python interpreter in the command prompt, you have been successful. Otherwise, an error similar to *python is not recognized as an internal or external command* will be displayed. To exit the Python interpreter, press $Ctrl + Z$ and then press *Enter*.

Python, NumPy and BioPython install in a very straightforward manner - just follow the instructions, accepting all defaults. Finally, there are three Python packages that need to be downloaded. *matplotlib* is relatively easy to install, again just follow the instructions provided on its website (listed below).
To install *acora*, you will need GCC or something akin to it for your platform. For Linux users, installation of GCC should be straightforward. For Windows users, something like MinGW is needed - I recommend installing the GCC Win32 binaries from `http://www.develer.com/oss/GccWinBinaries`, which is very straightforward to install - again, follow the instructions provided there. Installation on a Mac should be possible with Xcode, though I haven't tested this yet. Installation of the *Levenshtein* package will require Python *setuptools* `http://pypi.python.org/pypi/setuptools` to be downloaded and installed. As before, simply follow the instruction provided there to guide you through installation.

Once these have been installed, the following three Python packages can themselves be installed easily from the provided websites.

- matplotlib `http://sourceforge.net/projects/matplotlib/files/matplotlib/matplotlib-1.1.1/`
- acora `http://pypi.python.org/pypi/acora/1.7`
- Levenshtein `http://pypi.python.org/pypi/python-Levenshtein/`

Running Decombinator

Once the Decombinator folder has been downloaded and unpacked, Decombinator can be run from the command prompt as follows:

```
> cd home/yourname/Decombinator
> python DecombinatorV2_2.py -i YourSequences.fastq -o YourResults
```

This will create a results folder within your current directory (i.e. *home/yourname/Decombinator* in the example above) called *YourResults*. In this folder, four output files will be created, named *YourResults*, appended with one of the four (human) TcR chains (alpha, beta, gamma, delta). Each will contain the 5 variables form the identifier which uniquely categorises each distinct TcR sequence. The final 3 columns in the file will give the sequence id of each sequence successfully classified as a TcR, plus two columns containing NA (these are fields for a barcode, see below). Additionally, a file called *YourResults_summary* will be created which gives a summary of the analysis, including the number of sequences of each TcR chain found and the time taken amongst other statistics.

Further Functionality

```
> python DecombinatorV2_2.py -i YourSequences.fastq -o YourResults -b True -bs1 0 -be1 6 -bs2 6 -be2 12 -p True -c True -of True
-f True -ch all -s mouse
```

Further functionality is provided with additional arguments listed above, which are all *False* by default. The function of each of these arguments is listed below:

- -b, True or False, defines whether the TcR sequences contain barcodes in the first *bl* nucleotides.
- -bs1, an integer, defines the start position of the (first*) barcode in the nucleotide sequence.
- -be1, an integer, defines the end position of the (first*) barcode in the nucleotide sequence.
- -bs2, an integer, defines the start position of the (second*) barcode in the nucleotide sequence.
- -be2, an integer, defines the end position of the (second*) barcode in the nucleotide sequence.
- -p, True or False, defines whether to output plots of the analysis, and additional files detailing distinct clones and translated CDR3 amino acid sequences.
- -c, True or False, defines whether to include the count of each distinct clone in the output files of distinct clones if p is True.
- -of, True or False, defines whether to include out of frame sequences in the output files of translated sequences.

- -ch, all, alpha, beta, gamma, delta, defines whether to search for all TcRs (slow), or one particular chain (much faster).
- -f, True or False, defines whether to include the full TcR translated amino acid sequence (if True), or just the CDR3 sequence.
- -s, human or mouse, defines what species is under investigation.

* Note that if only one barcode is present then

`-be1`

and

`-bs2`

should be set to the same value, any integer between

`-bs1`

and

`-be2`

.

If sequence reads are much less than approximately 100 b.p, then it is recommended that you run a version of Decombinator that is specifically designed for short reads. To do this, include the argument

`-sh True`

.

Finally, two further scripts are provided as extras, which can be used to concatenate 2 paired-end read fastq files, to generate one fastq file with each sequence taking the 6 nucleotide barcode contained in one paired-end read and appending it to the sequence of the other. Usage is as follows:

```
> python AddAll2Hexr.py read1.fastq read2.fastq output.fq
```