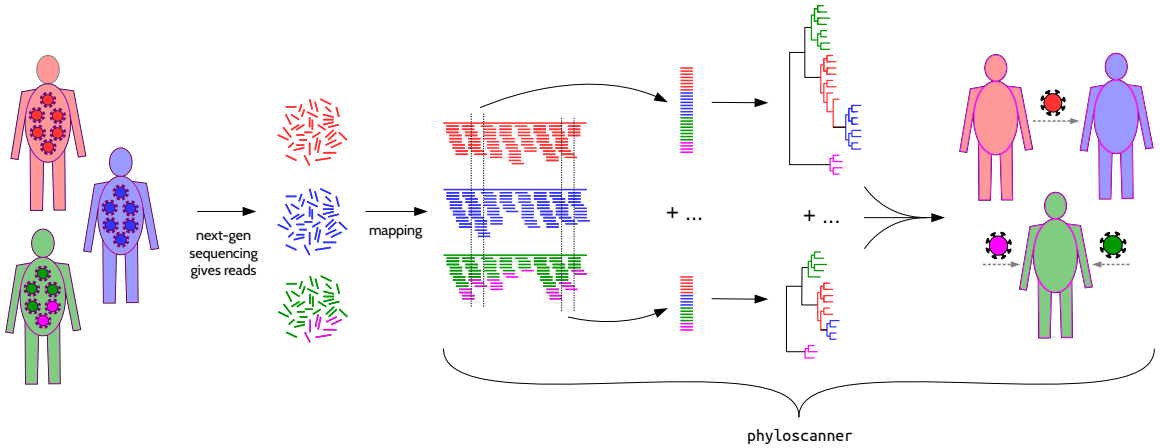
`phyloscanner`
available from github.com/BDI-pathogens/phyloscanner

Coding lead by Chris Wymant and Matthew Hall,
with contributions from Oliver Ratmann and Christophe Fraser

This manual last updated June 2, 2017



`phyloscanner` analyses pathogen genetic diversity and relationships between and within hosts at once, in windows along the genome using mapped *reads* (fragments of DNA produced by next-generation sequencing), or using phylogenies with multiple sequences per host previously generated by the user via any method. An example showing its action on simulated data (which is included with the code) can be seen at the GitHub repository linked to above.

If you have a problem getting the code to run or think you've found a bug, please create a *New issue* on the GitHub repository. The usual bug-reporting etiquette applies. (1) For non-trivial bugs, to allow us to reproduce the problem it's helpful to have input that results in the problem. This needn't be all your data. For example if one read in a bam file is causing a problem, you could make a new bam file containing only that read, allowing us to see the problem very quickly. Isolate the problem as much as you can first – help us to help you! (2) If you get an error when you run `phyloscanner` as part of a larger piece of code you've written yourself (e.g. a job submission script to a cluster), please don't make us debug that – the error might be in your code rather than ours. Isolate the problem to the relevant `phyloscanner` command.

Contents

Part I
Generating within- and between-host phylogenies in windows along the genome, using mapped reads (Chris)

1 The basic command

As input to `phyloscanner`, you need to have generated files of mapped reads in bam format – bam files henceforth. The bam format does not include the sequence to which the reads were mapped – the *reference* – which we also need. With the initial `$` traditionally indicating that what follows is a command to be run on the command line / in a terminal, the basic `phyloscanner` command looks like

```
$ phyloscanner.py ListOfMyInputFiles.csv --windows 1,300,301,600,...
```

where

- `ListOfMyInputFiles.csv` is a plain-text, comma-separated-variable (csv) format file in which the first column contains the bam files, the second column contains the corresponding reference files. An optional third column, if present, contains *aliases* – things to rename each bam file to in `phyloscanner` output; if not present the base name of the bam, i.e. the file name not including the directory path, will be used. e.g. This input file list might look like
  `PatientA.bam,PatientA_ref.fasta,A`
  `PatientB.bam,PatientB_ref.fasta,B`
  Bam file base names, and aliases if present, must be unique and free of whitespace. Quotation marks are interpreted as wrapping/protecting fields, allowing them to contain commas that are not field separators. You can generate this input file list however you like, including manually. The following block of code illustrates how you might generate it automatically from the command line. It also checks that your files exist, which `phyloscanner` does anyway, but errors are best caught soonest.

  ```
  $ for ID in PatientA PatientB PatientC; do
  $   bam=MyBamsDir/"$ID".bam
  $   ref=MyRefsDir/"$ID".fasta
  $   if [[ ! -f "$bam" ]]; then
  $     echo "$bam does not exist" >&2
  $     break
  $   elif [[ ! -f "$ref" ]]; then
  $     echo "$ref does not exist" >&2
  $     break
  $   fi
  $   echo "$bam","$ref","$ID"
  $ done > MyPhyloscannerInputFile.csv
  ```

  If those IDs were stored in a plain-text file `MyIDs.txt` separated by whitespace, you could replace the first line of that loop with
  `for ID in $(cat MyIDs.txt); do`
- The `--windows` (or `-W`) option is used to specify an even number of comma-separated positive integers: these are the coordinates of the windows to analyse, interpreted pairwise, i.e. the first two are the left and right edges of the first window, the third and fourth are the left and right edges of the second window, ... i.e. in the above example we have windows 1-300, 301-600, ...

Note that to run the `phyloscanner.py` binary file, like running any other binary file, you must either prepend to it the directory in which it lives (e.g. `/path/to/my/phyloscanner/code/phyloscanner.py`) or add that directory to the `$PATH` environment variable associated with your terminal (google this if needed).

2 What do the window coordinates mean exactly?

By default the references used for mapping, together with any extra references if included using the `--alignment-of-other-refs` option, are all aligned together and window coordinates are interpreted with respect to the alignment (i.e. position n refers to the nth column of that alignment, which could be a gap for some of the sequences). This alignment can be found in the file `RefsAln.fasta` after running `phyloscanner`, should you want to inspect it and possibly run again. You can manually specify window coordinates with respect to this alignment, using the `--windows` option, or have windows automatically chosen using `--auto-window-params`, which attempts to minimise the affect of insertions and deletions in the references on your window width and overlap preferences. Alternatively, if you are using the `--alignment-of-other-refs` option to include extra references, you can use `--pairwise-align-to` to name one of these references to be a kind of *reference reference*: instead of aligning of all the bam file references to each other, they will be sequentially and separately pairwise-aligned to your named reference, and window coordinates are interpreted with respect to that named reference. Using the `--pairwise-align-to` option is expected to more stable than `--windows` or `--auto-window-params` if your bam file references are many and diverse, since pairwise alignment is easier than multiple sequence alignment. It also has the advantage that when running `phyloscanner` more than once with different bam files, the coordinates mean the same thing each time.

3 What windows should I choose?

I'm glad you asked. It's important.

3.1 Start & end

You might as well fully cover the genomic region you're interested in. That requires choosing where to start and where to end. If you're interested in the whole genome, the start is 1 and the end is the genome length, or more precisely the length of the alignment of all references together (or the length of your named reference if you used the `--pairwise-align-to` option.) You may know that your reads don't start right at the beginning of the genome. If this is the case, a good place to start your first window would be the genome position at which you start having reads.

If your reads were generated by amplifying the sample using primers, the primers ought to have been trimmed from the reads as part of whatever bioinformatic pipeline produced your input bam files. (This can be done for example using `fastaq`, which is called as part of the `shiver` pipeline.) Then a sensible choice for the start of the first window would be the first position after the first primer, and a sensible choice for the end of the last window would be the last position before the last primer.

You might as well fully cover the genomic region you're interested in. Let's revisit that. In what part of the genome would you like to do phylogenetics? Perhaps all of it. Then again, remember that phylogenetics assumes neutral evolution at every included site. Though `phyloscanner` allows individual specified sites to be excised after alignment of all the reads from all the bam files, there may be regions of the genome where selected sites are so dense that it's simpler to skip the region altogether. There may also be regions of the genome where there are so many insertions and deletions (indels) that you're skeptical that the reads will align correctly in the first place. You can choose to specify no windows covering such regions.

3.2 Window width

As well as choosing where your first window should start and where your last one should end, you need to choose how wide each single window is. If a window is very small, so little diversity is contained inside it (within or between samples) that the number of *unique* reads overlapping the window is small, hindering meaningful phylogenetics. If window width exceeds read length, then you will have no reads in the window, since we keep only reads fully overlapping the window. Somewhere between these two extremes therefore maximises the number of unique reads; to help you figure out what that is for your data, you can run `phyloscanner` with the `--explore-window-widths` option. This reports, for each in a list of window widths to try, how many unique reads are found for each bam and at each position along the genome. To summarise these read counts into a single value that varies with window width, you could use the mean or the median, or you might be interested in a percentile lower than the 50th, if your

concern is ensuring some minimal amount of diversity across all bams and all genomic positions. Up to you.

(Note that some of the other options can affect how many reads you get in a window and so can affect what `--explore-window-widths` will tell you, namely the `--excision-coords`, `--merging-threshold`, `--min-read-count`, `--quality-trim-ends`, `--min-internal-quality`, and `--discard-improper-pairs` options. The first two can result in two or more unique reads being merged into one; the rest can simply discard some reads. You could choose values for the associated parameters immediately and then use `--explore-window-widths`, or else come back to `--explore-window-widths` later on once you've got the hang of `phyloscanner` and investigated the effect of those other options in your data.)

Power users might want to optimise their own measure of phylogenetic information as a function of window width; one of the first metrics to pop into your head might be the mean bootstrap of all nodes in the tree. That's not advised because within a sample there may be many very similar sequences, and the set of nodes connecting these may have poor boostrap support, but this is not something that ought to be penalised. Also in theory you might be able to increase the window width until only a single read is found spanning the window in each patient; your bootstraps might then be great, because between-host diversity is greater than that within-host, but you've thrown out all the within-host information.

NB wherever *read* and *read length* appeared in the discussion above, they should be substituted for *insert* and *insert size* if you have paired-read data AND the reads in a pair sometimes overlap AND you run `phyloscanner` with `--merge-paired-reads` to merge overlapping paired reads into a single longer read (see the cartoon below). A complication with this is that whereas read length is typically fixed within a sample, insert size has a distribution of different values. A window which is wider than twice the read length can never get any reads, because the reads in a pair need to overlap in order to be merged. So you have two choices.

1. Choose
   (read length) $<$ (window width) $<$ (twice the read length)
   Then you're restricted to the subset of read pairs that satisfy (window width) $\leq$ (insert size) $<$ (twice the read length) because only such pairs can overlap and fully span the window. The fraction of such reads in a sample is the integral of the unit-normalised insert size distribution between the two limits in the inequality above.

2. Choose
   (window width) $\leq$ (read length)
   Then you can have single reads contribute in addition to merged overlapping read pairs. But perhaps that window is too short; see the discussion on window width above.

3.3 Window overlap

How much should neighbouring windows overlap? A simple answer to this is zero, i.e. each window starts right after the previous one ends. e.g. 1-99, 100-199, 200-299, ...
TODO: discuss positive overlap.

4 Optional arguments

Part II

Analysing within- and between-host phylogenies (Matthew)