A brief tutorial on SEPP

Siavash Mirarab

January 31, 2013

Contents

1 Introduction to SEPP

SEPP stands for SATé-Enabled Phylogenetic Placement and addresses the problem of phylogenetic placement for meta-genomic short reads. More precisely, SEPP addresses the following problem.

Input: tree T and alignment A for a set of full-length gene sequences, and set X of fragmentary sequences for the same gene

Output: placement of each fragment in X into the tree T, and alignment of each fragment in X to the alignment A.

Phylogenetic placement puts unknown short fragments in a phylogenetic context, and hence helps identifying species included in a metagenomic dataset. Phylogenetic placement involves two steps: alignment of short fragments to full length sequence alignment A (also called the *backbone* alignment), and then placement of aligned short reads on the fixed tree T (also called the *backbone* tree).

SEPP operates by using a divide-and-conquer strategy adopted from SATé [**?**] to improve alignment of fragments (produced by running HMMER ([**?**]). It then places each fragment into the user-provided tree using pplacer ([**?**]). Our study shows that SEPP provides improved accuracy for quickly evolving genes as compared to other methods. For more information refer to our publish paper (`http://www.ncbi.nlm.nih.gov/pubmed/22174280`)

2 Installing SEPP

First you are going to setup SEPP on your machines. SEPP currently runs only on Linux and Mac. Those running a Windows need to either use cygwin (`http://www.cygwin.com/`), or a Ubuntu virtual machine image we have created.

To download and install SEPP, follow these steps:

1. Download software from `https://github.com/smirarab/sepp/zipball/master` (or copy it from the memory sticks provided to you).
2. Copy the archive to your favorite location, and unpack the zip file using your favorite software.
3. Open a terminal and change into the unpacked directory.
4. In the new directory, there should be a setup.py file. Run the following command:

   ```
   sudo python setup.py install
   ```

   If you don't have root access, instead use the following command (or use `--prefix` option, as described in the README file)

   ```
   python setup.py install --user
   ```

Refer to the README file for more information regarding the installation, and solutions to common installation problems.

2.1 Using Virtual Machine

An Ubuntu VM image with SEPP installed on it is provided to participants, and is available for download at `http://www.cs.utexas.edu/~phylo/software/sepp/`.

If you were unable to install SEPP on your machine, you can use the VM image provided to you by organizers. You first need to copy the VM image to your machine. Then, open the VM image in your favorite VM software (e.g. VirtualBox) and start the VM. Once your VM starts, you can run SEPP from a terminal.

3 Running SEPP

In this section we will run SEPP on a sample dataset provided with the software. SEPP is currently available only as a command line tool.

3.1 Run SEPP with -h option to see the help

- Make sure you have a terminal open.
- Make a directory where you will run SEPP and cd into it (e.g. `mkdir seppRuns; cd seppRuns;`)
- run SEPP with -h option to see a help:
  `run_sepp.py -h`

3.2 Sample Datasets - Default parameters

- Copy test datasets into your directory. Test datasets can be found under sepp/test/testdata under the directory where you unpacked SEPP archive. On the vital machine, you can find test files at $HOME/SEPP/sepp/test/testdata.

- Execute the following command to run SEPP on a sample biological dataset.

  ```
  run_sepp.py -t mock/pyrg/sate.tre -r mock/pyrg/sate.tre.RAxML_info -a
  mock/pyrg/sate.fasta -f mock/pyrg/pyrg.even.fas
  ```

  (SEPP should finish running in about 2 minutes.)

This runs SEPP on the given example dataset. All the options provided to SEPP in this example run are mandatory. As you can see, there are few inputs required to run SEPP. The following describes the minimum input of SEPP:

Backbone tree is the tree on which SEPP places short fragments. This tree should be a ML tree, estimated using RAxML [**?**] or phyml [**?**][1] based on the backbone alignment (see below). The input tree is given to SEPP using -t option, and needs to be in newick format (as outputted by RAxML for example).

Backbone Alignment is a multiple sequence alignment of full length sequences. Full length sequences and the placed fragments should all come from the same gene, and placement usually makes sense only if the identity of the full length sequences is known. Backbone alignment is used as a reference for alignment of fragments. The backbone alignment and tree should be on the same exact set of taxa, and backbone tree should be estimated based on the backbone alignment. Backbone alignment is provided to SEPP using -a option, and should be in Fasta format (refseq available at `http://www.ncbi.nlm.nih.gov/RefSeq/` is a nice tool for conversion between different alignment formats).

Stats or info file is the info file generated by RAxML (or phyml) when backbone tree was being estimated from the backbone alignment. This file is required by pplacer (run internally by SEPP) in order to avoid re-estimation of ML parameters. To be able to use SEPP you need to make sure you keep your info file when you are generating the backbone tree. If you do not have the info file (or if you used other software programs such as SATé to get your trees), you can always use RAxML's `-f e` option to quickly estimate model parameters (including branch lengths) on your fixed tree topology. RAxML info file should be provided to SEPP using -r option.

Fragments file is a Fasta file containing the actual short fragments that are going to be placed. Fragments file should be given to SEPP using -f option.

[1] we haven't tested SEPP with phyml trees yet.

3.2.1 10% rule

Recall that SEPP operates by dividing backbone alignment and tree into alignment subsets and placement subsets. In the above run, we did not explicitly set the subset sizes we desire for alignment and placement subsets. Choice of these two parameters affects accuracy on one hand, and computational resources on the other hand, as explained below.

- Decreasing alignment sizes should increase (and have increased in our experience) the accuracy of SEPP. On the other hand, smaller subsets increase the running time. This is because SEPP needs to score each fragment against all subsets independently, and therefore increasing the number of subsets adds to the running time. Note that extremely small subsets (i.e. less than 5 taxa) have not been tested extensively and are not recommended.

- Increasing placement sizes should result in better accuracy in general (although there could be exceptions). If your placement tree is very large (thousands or tens of thousands of leaves) the memory requirement of pplacer, and hence SEPP increases dramatically. Reducing the placement size reduces the memory footprint, and hence enables placement on larger trees given a fixed amount of memory available. This would be one of the main motivations to reduce placement subset size. Reducing the placement subset *can* result in reduced running time as well, especially if your placement tree has thousands of taxa. For smaller trees, the effect of the placement size on the running time is not easily predicted, and is practically of less interest.

By default, when alignment and placement subset sizes are not explicitly specified by user, SEPP uses what we call the 10% rule to automatically set those parameters. 10% rule specifies that alignment and placement subset sizes should be both set to 10% of the number of full length sequences (i.e. number of leaves in the backbone tree). The 10% rule is just a heuristic setting we have found empirically to give a reasonable tradeoff *in general* between accuracy and computational requirements on the datasets we have tried. Users are encouraged to change subsets sizes based on their available computational resources, and the desired accuracy, according to the guidelines outlined above.

3.2.2 Specifying subset sizes using -A and -P options

Imagine we cannot wait 2 minutes to get results on our test dataset. We are going to increase the alignment subset so that SEPP runs faster. The test dataset included 65 full length sequences, and hence 10% rule amounts to alignment and placement subsets of size maximum 7. Since our toy example of 65 sequences is very small, it makes sense to increase the alignment size to a larger number (e.g. 10), and placement size to the entire dataset (i.e. 65). Maximum alignment and placement subset sizes are controlled with -A an -P options respectively.

- Execute the following command on your terminal to run SEPP with -A=10 and -P=65

  ```
  run_sepp.py -t mock/pyrg/sate.tre -r mock/pyrg/sate.tre.RAxML_info -a
  mock/pyrg/sate.fasta -f mock/pyrg/pyrg.even.fas -A 10 -P 65 -o run2.A10P65.
  ```

  (SEPP should finish in about 1 minute.)

In the above run note the -o option. This option controls the prefix of the output files generated by SEPP. We have not looked at SEPP output yet, and we will do so in a moment. For now, just be aware that SEPP generates a bunch of output files, and prefixes those with a given string, which defaults to "output". These outputs are by default generated in the current directory, but that can be changed using the -d option. Had we not changed the output prefix, SEPP would have refused to run to avoid overwriting results of your previous run saved with "output" prefix. Try this, and you would get the following error:
`directory [.] already contains files with prefix [output]... Terminating to avoid loss of existing files.`

3.3 Running SEPP on a larger dataset

Under mock/rpsS folder there is another dataset with more than 1,200 full length sequences and 2000 fragments. We are going to start a SEPP run on this dataset (please skip this if you are sharing a server with other users and are short on memory). Run the following command:

```
run_sepp.py -t mock/rpsS/sate.tre -r mock/rpsS/sate.tre.RAxML_info
-a mock/rpsS/sate.fasta -f mock/rpsS/rpsS.even.fas -o rpsS.out.default
```

This run is going to take about 4 minutes. While this is running, we are going to look at SEPP outputs.

3.4 SEPP output

SEPP has two outputs: an alignment of fragments to full length sequences (or subsets of the full length sequences), and placements of fragments on the given backbone tree. When SEPP finishes running, it generates two or more output files, all prefixed with a string given using -o option, and placed in a directory given using -d option. One of these output files is [prefix].json file. This is the results of the placement algorithm, and is in a format devised by the pplacer software. This .json file is a human-readable text file. However in most cases you want to look at those results in a visualization tool. pplacer comes with a suit of software tools called guppy. guppy reads a .json file, and among other things, produced nice visualizations of the results. We are going to first

manually look at the plain .json file to understand its content, and then will use guppy to visualize results.

- Open the file called output.json using a text editor.
- Notice at the top of the file there is a newick tree with edges labeled with numbers inside brackets.
- Following the tree, placement results are given for each fragment. Everything between { and } marks describes the placement of a single fragment. Each fragment can have multiple placements, with different likelihoods. Each line under the "p" attribute indicates one placement of the fragment. The first value gives the edge label, the second value gives the log likelihood, the third value gives probability of that placement, and the final two values give the position on the edge where the fragment is placed, and the length of the pendant edge for the fragment.

Next we will use guppy to turn this text file to a visualization of the results. Issue the following command to generate a tree that has fragments placed on the backbone tree based only on *the best placement* of each fragment.

```
guppy tog --xml output.json
```

This command generates a new file called output.tog.xml. This is an XML file in NexML format (`http://www.nexml.org/`) and can be opened with Archaeopteryx software available at `http://www.phylosoft.org/archaeopteryx/`. You can find Archaeopteryx also on the USB drive provided to you by organizers of the workshop. Run Archaeopteryx and use `File/Read Tree from File` to open output.tog.xml. Select "colorize branch color" checkbox on the right hand side panel. The white branches represent the backbone tree, and branches in red correspond to the maximum likelihood placement of fragments on the backbone tree.

By now the run we started on the larger dataset (rpsS gene) should have finished as well (it takes about 5 minutes). You can use guppy to visualize the results of that run as well.

Please refer to pplacer and guppy documentation at `http://matsen.github.com/pplacer/generated_rst/pplacer.html` for more information about .json format and visualization options available in guppy.

In addition to the placement file, one or more alignment files are also generated. These alignment files show how fragments are aligned to the backbone alignment (or subsets of the backbone alignment). These are simple Fasta files, and can be viewed in any alignment visualization tool (e.g. JalView available at `http://www.jalview.org/`).