A brief tutorial on SEPP

Siavash Mirarab, Nam Nguyen, Tandy Warnow

February 11, 2013

Contents

1 Introduction to SEPP

SEPP stands for SATé-Enabled Phylogenetic Placement [7] and addresses the problem of phylogenetic placement for meta-genomic short reads. More precisely, SEPP addresses the following problem.

Input: *backbone* 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 into 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 [4, 5] to improve the alignment of fragments to the backbone alignment (produced by running HMMER [1]). It then places each fragment into the user-provided tree using pplacer [6]. Our study shows that SEPP provides improved accuracy for quickly evolving genes as compared to other methods. For more information see [7] (available at: `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.

2.1 Installing on a Linux Machine

To download and install SEPP, follow these steps:

1. Download software from `https://github.com/smirarab/sepp/zipball/master`.
2. Copy the archive to your favorite location (we will use ~/sepp in this tutorial), and unpack the zip file using your favorite software.
3. Open a terminal and change into the unpacked directory (`cd ~/sepp`).
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
   ```

5. Run the following command:

   ```
   python setup.py config
   ```

   Note that while step 1-4 can be done once per machine (if installed with root access), step 5 needs to be run by each individual user. This step creates a

.sepp/main.config file in your home directory and copies the bundled tool under the same directory. This file contains defaults settings of sepp, and can be modified by users if needed.

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

2.2 Using Virtual Machine

An Ubuntu VM image with SEPP installed on it 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 this VM image. 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. The username and password for the virtual machine are "ubuntu" and "reverse".

3 Running SEPP

SEPP is currently available only as a command line tool, and so the tutorial is based on this command line usage.

In this section we will run SEPP on a small sample dataset provided with the software. The sample dataset consists of a SATé backbone alignment and tree on the "pyrg" marker gene with only 65 sequences (previously studied in [3]). The fragments are from a WGS sample of a mock community created by the NIH Human Microbiome Project (`http://www.hmpdacc.org/HMMC/`). The fragment file we provide includes only 106 fragments that we found to possibly belong to "pyrg" marker (based on hits to its HMMER profile).

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 are part of the distribution and can be found under ~/sepp/test/unittest/data (use the directory where you

unpacked SEPP instead of ~/sepp). For example, on Unix run:
`cp -R ~/sepp/test/unittest/data/* .`

- 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: this is the tree on which SEPP places short fragments. This tree should be a binary maximum likelihood (ML) tree in newick format; we therefore recommend you estimate the ML tree using RAxML [8] or phyml [2][1] on the backbone alignment (see below). The input tree is given to SEPP using -t option.

Backbone Alignment: this is a multiple sequence alignment of full length sequences for some gene. These sequences need to be for the same gene as the fragmentary sequences, as phylogenetic placemeent only makes sense in this case. The backbone alignment needs to be highly accurate, since it determines the backbone tree, and the use of RAxML to estimate an ML tree on the backbone alignment may help with the accuracy. If you obtain a backbone tree in some other way, you should ensure that the backbone tree and alignment are on the same exact set of taxa. The backbone alignment is provided to SEPP using -a option, and should be in the Fasta format. You can use refseq, available at `http://www.ncbi.nlm.nih.gov/RefSeq/`, to convert between different alignment formats).

Stats or info file: this is the info file generated by RAxML (or phyml) when it computes the ML tree on the backbone alignment (i.e., the backbone tree). 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 some other software programs, such as SATé, to produce the backbone tree), you can use RAxML's `-f e` option to quickly estimate the model parameters (including branch lengths) on your backbone tree topology (see **??** for details). The RAxML info file should be provided to SEPP using -r option.

Fragments file: this 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]Note - we haven't tested SEPP with phyml trees yet, and all our analyses are based on RAxML.

3.2.1 10% rule

Recall that SEPP operates by dividing the set of taxa in the backbone alignment and tree into alignment subsets and placement subsets. In the above run, we did not explicitly set the maximum subset sizes for alignment and placement subsets. The choice of these two parameters affects accuracy on the 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 of 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 the 10% rule amounts to alignment and placement subsets of maximum size 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). The maximum alignment and placement subset sizes are controlled with -A and -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:

```
Output directory [a_drecotry] already contains files with prefix [output]...
Terminating to avoid loss of existing files.
```

3.3 Running SEPP on a larger dataset

We are now going to run SEPP on a larger dataset. Similar to the small "pyrg" dataset, our larger dataset is on fragments from a WGS sample of the HMMC mock community (`http://www.hmpdacc.org/HMMC/`), but is based on a much larger marker gene called "rpsS" (obtained from [3]). The backbone alignment and tree are again estimated using SATé. This dataset is available under mock/rpsS folder and has 1,277 full length sequences and 2101 fragments. We are going to start a SEPP run on this dataset. 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.

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]_placement.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_placement.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 "}" 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 (as estimated using the maximum likelihood calculation in pplacer) 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.

```
~/.sepp/bundled-v2.2/guppy tog --xml output_placement.json
```

This command generates a new file called out_placement.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/`. Run Archaeopteryx and use `File/Read Tree from File` to open out_placement.tog.xml. Select "colorize branch" 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). 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, an extended alignment file is also generated. This extended alignment shows how fragments are aligned to the backbone alignment. The extended alignment is a simple Fasta file, and can be viewed in any alignment visualization tool (e.g. JalView available at `http://www.jalview.org/`).

5 SEPP Obtaining Backbone Alignment and Trees

In all examples given above, backbone alignment and trees were already estimated. Our suggested way of obtaining backbone alignment and trees is through SATé [5], which simultaneously estimates both an alignment and a tree based on unaligned full length sequences. Please refer to SATé documentation for more information on running SATé.

In addition to an alignment and a tree (obtained from SATé or otherwise), we also need to have a RAxML info file. If your backbone tree is estimated using RAxML you already have the info file. Otherwise, you can optimize model parameters on your backbone tree by running the following:

```
raxml -f e -t [backbone tree] -s [backbone alignment] -m GTRGAMMA -n [a name]
```

This will optimize GTRGAMMA model parameters on your input alignment/tree pair and will generate a info file (RAxML_info.a_name), that can be used with SEPP.

6 SEPP Miscellaneous

The following are some other points that are worth mentioning and testing.

- By default, the input is assumed to be DNA. To analyze amino acid datasets use `-m` option (i.e. `-m amino` for amino acid and `-m rna` for RNA).
- By default SEPP tries to use all available cores on your machine in each run. If you run multiple instances of SEPP simultaneously, or if you want it to use fewer cores, be sure to set the number of cpus used by SEPP using `-x` option. For example the following runs SEPP on the large dataset but with only 3 cpus used:

  ```
  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 -x 3
  ```

- In addition to commandline, SEPP can be controlled through a configuration file, passed to SEPP using `-c` option. For example, to run SEPP using config.run1 configuration file, use:

  ```
  run_sepp.py -c config.run1
  ```

  Commandline options can be specified in the configuration file under the `[commandline]` section. For specifying options under `[commandline]`, you need to use their long format name (as show in the SEPP help invoked by `-h`). For example, to set input to "rpsS" dataset and the alignment size to 10 and number of cpus to 3 use:

```
[commandline]
alignmentSize = 10
tree= mock/rpsS/sate.tre
raxml = mock/rpsS/sate.tre.RAxML_info
alignment = mock/rpsS/sate.fasta
fragment = mock/rpsS/rpsS.even.fas
cpu = 3
```

Some extra information (not available in commandline) can also be configured in other sections of the configuration file. For example,

```
[pplacer]
path = /some/path
```

tells SEPP that pplacer binaries can be found under /some/path.

An example config file is available as part of the distribution under the test directory (test/unittest/data/simulated/sample.config). A main configuration file under {home}/.sepp/main.config is used to store some basic configurations such as the location of extra programs, etc. When conflicting options are given, precedence is with those provided through commandline, then those specified in config file provided using `-c` option and finally those specified in the main config file.

- Currently SEPP has a built-in checkpointing functionality. By default, this functionality is turned off, but can be turned on using `-cp [checkpoint file name]`, and the time interval between two consecutive checkpoints can be adjusted using `-cpi [checkpoint frequency in seconds]`. Please note that checkpointing options have been tested only lightly and might have unknown issues.

References

[1] Sean R Eddy. Profile hidden Markov models. *Bioinformatics*, 14(9):755–763, 1998.

[2] Stéphane Guindon, Jean-François Dufayard, Vincent Lefort, Maria Anisimova, Wim Hordijk, and Olivier Gascuel. New algorithms and methods to estimate Maximum-Likelihood phylogenies: Assessing the performance of PhyML 3.0. *Systematic Biology*, 59(3):307–321, May 2010.

[3] Bo Liu, Theodore Gibbons, Mohammad Ghodsi, and Mihai Pop. MetaPhyler: Taxonomic profiling for metagenomic sequences. In *Bioinformatics and Biomedicine (BIBM), 2010 IEEE International Conference on*, pages 95–100. IEEE, 2011.

[4] Kevin Liu, Sindhu Raghavan, Serita Nelesen, C. Randal Linder, and Tandy Warnow. Rapid and Accurate Large-Scale Coestimation of Sequence Alignments and Phylogenetic Trees. *Science*, 324(5934):1561–1564, June 2009.

[5] Kevin Liu, Tandy J Warnow, Mark T Holder, Serita M Nelesen, Jiaye Yu, Alexandros P Stamatakis, and C Randal Linder. SATe-II: Very Fast and Accurate Simultaneous Estimation of Multiple Sequence Alignments and Phylogenetic Trees. *Systematic Biology*, 61(1):90–106, 2011.

[6] Frederick Matsen, Robin Kodner, and E. Virginia Armbrust. pplacer: linear time maximum-likelihood and Bayesian phylogenetic placement of sequences onto a fixed reference tree. *BMC Bioinformatics*, 11(1):538+, October 2010.

[7] Siavash Mirarab, Nam Nguyen, and Tandy Warnow. SEPP: SATé-Enabled Phylogenetic Placement. *Pacific Symposium On Biocomputing*, pages 247–58, 2012.

[8] Alexandros Stamatakis. RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, 22(21):2688–2690, 2006.