DESCRIPTION:

ASTRAL is a Java program for estimating a species tree given a set of unrooted gene trees. ASTRAL is statistically consistent under multi-species coalescent model (and thus is useful for handling ILS). It finds the tree that maximizes the number of induced quartet trees in the set of gene trees that are shared by the species tree. The algorithm has an exact version that can run for small datasets (less than 18 taxa) and a more useful version that can handle large datasets (103 taxa and 800 genes were analyzed in few minutes).

The algorithm used is described in:

a. Mirarab, R. Reaz, Md. S. Bayzid, T. Zimmermann, M.S. Swenson, and T. Warnow1 "ASTRAL: Genome-Scale Coalescent-Based Species Tree Estimation", accepted in ECCB 2014 and to appear in Bioinformatics

Email: `astral-users@googlegroups.com` for questions

Tutorial Steps:

Step 1: INSTALLATION:

There is no installation required to run ASTRAL. You simply need to download the zip file and extract the contents to a folder of your choice. Alternatively, you can clone the github repository if you are familiar with git. If you clone the git repository, you can run `make.sh` to build the project, or simply use the jar file that is included with the repository.

ASTRAL is a java-based application, and should run in any environment (Windows, Linux, Mac, etc.) as long as java is installed. Java 1.5 or later is required. We have tested ASTRAL only on Linux and MAC, but it should work on

Windows too.

In the remaining of the tutorial, we will assume you have the extracted ATRAL zip file into a directory called `~/astral-home/`. In the commands given below, substitute `~/astral-home/` with the directory you have chosen for ASTRAL.

Step 2: Running ASTRAL from command-line to see the help:

ASTRAL currently has no GUI. You need to run it through command-line. Open a terminal (in Windows, look for a program called `Command Prompt` and run that; In Linux you should know how to do this; In MAC, search for an application called `Terminal`). Once the terminal is opened up, go the location where you have downloaded the software (e.g. using `cd ~/astral-home/`), and issue the following command:

```
   java -jar astral.4.6.0.jar
```

This will print the list of options available in ASTRAL. If no errors are printed, your ASTRAL installation is fine and you can proceed to the next steps. Let us know if you get an error message at this point.

Step 3: Running ASTRAL on a sample input dataset

We will next run ASTRAL on an input dataset. From the ASTRAL directory, run:

```
java -jar astral.4.6.0.jar -i test_data/song_mammals.424.gene.
tre
```

The results will be outputted to the standard output. To save the results in an output file use the `-o` option:

```
java -jar astral.4.6.0.jar -i test_data/song_mammals.424.gene.
tre -o test_data/song_mammals.tre
```

Here, the main input is just a file that contains all the input gene trees in newick format. The input gene trees are treated as unrooted, whether or not they have a root. Note that the output of ASTRAL should also be treated as an unrooted tree.

The test file that we are providing here is based on the Song et. al. dataset of 37 mammalian species and 442 genes. We have removed 23 problematic genes (21 mislabeled genes and 2 genes we classified as outliers) and we have also re-estimated gene trees using RAxML on the alignments that authors of that paper kindly provided to us.

Some points regarding the input file are worth mentioning upfront:

Polytomies: The input gene trees can have polytomies (unresolved branches) since version 4.6.0.

Multiple Indiviuals: ASTRAL can handling multiple individuals from the same species since version 4.5.0. If `s1i1` , `s1i2` , `s1i3` all belong to a species called `S1` , and similarly, `s2i1` , `s2i2` all belong to a species called `S2` , a mapping file with one of the following formats needs to be provided to ASTRAL:

```
S1 3 s1i1 s1i2 s1i3
S2 2 s2i1 s2i2
```

or

```
S1:s1i1,s1i2,s1i3
S2:s2i1,s2i2
```

Internal node labels: Since version 4.2.1, ASTRAL can handle internal node labels.

Step 4: Viewing results of ASTRAL:

The output of ASTRAL is a tree in newick format. These trees can be viewed in many many existing tools. Here are few that are used by many people:

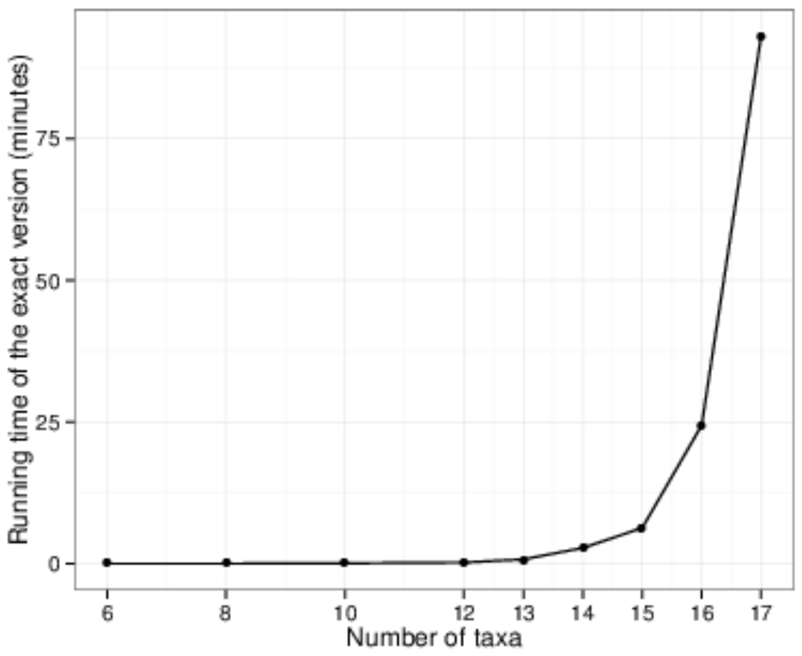
1. FigTree is probably the most widely used tool. It produces nice looking figures and works under Linux, Windows and MAC.
2. Archaeopteryx is very good for viewing large trees and also works under all three operating systems.
3. EvolView: online application. You don't even need to download.

There are many more.

For this tutorial, let's use the online viewer (EvolView) or any other tool you can manage to download and install. Using either of these applications open the `test_data/song_mammals.tre` file. We will explore various tree viewing options. Importantly, we will reroot the tree at the correct node, which is always necessary, since the rooting of the ASTRAL trees is generally arbitrary and meaningless.

Step 5: The exact version of ASTRAL

ASTRAL has an exact and a heuristic version. The heuristic version solves the optimization problem exactly subject to the constraint that all the bipartitions in the species tree should be present in at least one of the input gene trees, or in a set of few other bipartitions that (since version 4.5.1) ASTRAL automatically infers from the set of input gene trees as likely bipartitions in the specie tree. The constrained version of ASTRAL is the default. However, when you have small number of taxa (typically 17 taxa or less), the exact version of ASTRAL can also run in reasonable time (see the figure below).



Since the mammalian dataset we have used so far has 37 taxa, the exact version cannot run on it. However, we have created a subset of this dataset that has all 9 primates, tree shrew, rat, rabbit, horse, and the sloth (a total of 14 taxa). We can run the exact version of ASTRAL on this reduced dataset. Run:

```
java -jar astral.4.6.0.jar -i test_data/song_primates.424.gene
.tre -o test_data/song_primates.424.exact.tre -x
```

Using the `-x` option results in running the exact version of the ASTRAL algorithm. This run should finish in about 30 seconds. Now, we will run ATRAL on the same input using the default heuristic algorithm:

```
java -jar astral.4.6.0.jar -i test_data/song_primates.424.gene
.tre -o test_data/song_primates.424.default.tre
```

This time, ASTRAL finished in under a second. So, is there a difference between the output of the exact and the heuristic version? Open up the two trees in your tree viewer tool and compare them. You will notice they are identical. You could also compare the scores outputted by ASTRAL and notice that they are identical.

Step 6: Reduced number of gene trees:

The default primate dataset we used in the previous step had 424 genes and 14 taxa. Since we have a relatively large number of gene trees, we could reasonably expect the exact and heuristic versions to generate identical output. The key point here is that as the number of genes increases, the probability that each bipartition of the species tree appears in at least one input gene tree increases. Thus, with 424 genes all bipartitions from the species tree are in at least one input gene tree, and therefore the exact and the heuristic versions are identical. We will demonstrate this point to you through an example.

We tried hard to find a subset of genes in the biological primates dataset where the exact and the heuristic versions did not match. We couldn't! So we had to resort to simulations. We simulated a primate-like dataset but we increased the amount of ILS by a large margin (divided all branch lengths by 5). Now, with this simulated primate dataset, if you take only 10 genes, something interesting happens:

```
java -jar astral.4.6.0.jar -i test_data/simulated_primates_5X.
10.gene.tre -o test_data/simulated_primates_5X.10.species.defa
ult.tre
```

and then

```
java -jar astral.4.6.0.jar -i test_data/simulated_primates_5X.
10.gene.tre -o test_data/simulated_primates_5X.10.species.exac
t.tre -x
```

These two commands will run the default an the exact versions of the ASTRAL algorithm on a 10 genes 14 taxa simulated primate dataset with extreme ILS.

NOTE: Since version 4.5.1, the heuristic and exact versions perform identically for our test dataset (because the heuristic version has improved to look at extra bipartitions). We need to find a new test case where exact and heuristic are actually different (not an easy task by any means).

Imagine that for a better test case that we are struggling to find, the tree outputted by the exact version has a slightly higher score, and a slightly different tree compared to the heuristic version. We would then say, in extreme cases (i.e. lots of ILS and/or gene tree estimation error and few available gene trees compared to the number of taxa), one could observe differences between exact and heuristic versions. Finally, note that how many genes should be considered few depends on the number of taxa you have, and also how much missing data there is.

Step 7: Providing ASTRAL with extra trees:

What if we had few very discordant gene trees, but the number of taxa of sufficiently large that we couldn't run the exact version? We always have a second option. Imagine that you are able to create a set of hypothetical trees using various methods. For example, maybe you have prior hypothesis of what the species tee should be. Or, maybe you have run concatenation and have potential species trees. Most realistically, maybe you have a collection of bootstrapped gene tress that can be used. ASTRAL allows you to provide these sets of alternative trees to expand the space of that ASTRAL considers. Thus, ASTRAL will solve the optimization problem subject to the constraint that each bipartition should come either from one of the input gene trees, or the ones it infers automatically, or these "extra" gene trees. The extra gene trees, however, do not contribute to the score. They just control the space bing searched.

So, now run:

```
java -jar astral.4.6.0.jar -i test_data/simulated_primates_5X.10.gene.tre -o test_data/simulated_primates_5X.10.species.tre -e test_data/simulated_primates_5X.10.bootstrap.gene.tre
```

Here, the `-e` option is used to input a set of extra trees that ASTRAL uses to expand its search space. The file provided simply has 200 bootstrap replicates for each of the these 10 simulated genes. Note that with this extra file as input, now ASTRAL is able to find the same results as the exact version (again, we need to find a better example, because since version 4.5.1 ASTRAL is identical on exact and heuristic versions to begin with).

Step 8: Running on large datasets:

We have now finished looking at all the important options of ASTRAL (you can ignore the rest). To finish, we will run ASTRAL on a relatively large dataset. Run:

```
java -jar astral.4.6.0.jar -i test_data/100-simulated-boot
```

The input file here is a simulated dataset with 100 sequences and 100 replicates of bootstrapped gene trees for 25 loci (thus 2,500 input trees). Note that ATRAL finishes on this dataset in a matter of seconds.

Step 9: Multi-locus Bootstrapping:

Astral can perform multi-locus bootstrapping (Seo, 2008). To be able to perform multi-locus bootstrapping, ASTRAL needs to have access to bootstrap replicates for each gene. To start multi-locus bootstrapping using ASTRAL, you need to provide the location of all gene tree bootstrap replicates. To run bootstrapping on our test input files, go to `test_data` directory, and decompress the file called `song_mammals.424genes.bs-trees.zip`. Now run

```
java -jar ../astral.4.6.0.jar -i song_mammals.424.gene.tre -b
bs-files
```

This will run 100 replicates of bootstrapping. The argument after `-i` (here `song_mammals.424.gene.tre`) contains all the maximum likelihood gene trees (just like the case where bootstrapping was not used). The `-b` option tells ASTRAL that bootstrapping needs to be performed. Following `-b` is the name of a file (here `bs-files`) that contains the location of gene tree bootstrap files, one line per gene. For example, the first line is `424genes/100/raxmlboot.gtrgamma/RAxML_bootstrap.allbs`, which tells ASTRAL that the gene tree bootstrap replicates of the first gene can be found in a file called `424genes/100/raxmlboot.gtrgamma/RAxML_bootstrap.allbs`.

By default ASTRAL performs 100 bootstrap replicates, but the `-r` option can be used to perform any number of replicates. For example,

```
java -jar ../astral.4.6.0.jar -i song_mammals.424.gene.tre -b
bs-files -r 150
```

will do 150 replicates. Note that your input gene tree bootstrap files need to have enough bootstrap replicates for the number of replicates requested using `-r`. For example, if you have `-r 150`, each file listed in `bs-files` should contain at least 150 bootstrap replicates.

Also, ASTRAL performs site-only resampling by default (see Seo, 2008). ASTRAL can also perform gene/site resampling, which can be activated with the `-g` option:

```
java -jar ../astral.4.6.0.jar -i song_mammals.424.gene.tre -b
bs-files -g -r 100
```

Note that when you perform gene/site resampling, you need more gene tree replicates than the number of multi-locus bootstrapping replicates you requested using `-r`. For example, if you have `-g -r 100`, you might need 150 replicates for some genes (and less than 100 replicates for other genes). This is because when genes are resampled, some genes will be sampled more often than others by chance.

Finally, since bootstrapping involves a random process, a seed number can be provided to ASTRAL to ensure reproducibility. The seed number can be set using the `-s` option (by default 692 is used).

Output: ASTRAL first performs the bootstrapping, and for each replicate, it outputs the bootstrapped ASTRAL tree. So, if number of replicates is set to 100, it outputs 100 trees. Then, it outputs a greedy consensus of all the 100 bootstrapped trees (with support drawn on branches). Finally, it performs the main analysis (i.e. on trees provided using `-i` option) and draws branch support on this main tree using the bootstrap replicates. The tree outputted at the end therefore is the ASTRAL tree on main input trees, with support values drawn based on bootstrap replicates. Support values are shown as branch length (i.e. after a colon sign) and are percentages.

Miscellaneous :

Memory:

For big datasets (say more than 100 taxon) increasing the memory available to Java can result in speed up. Note that you should give Java only as much as free memory you have available on your machine. So, for example, if you have 3GB of free memory, you can invoke ASTRAL using the following command to make all the 3GB available to Java:

```
java -Xmx3000M -jar astral.4.6.0.jar -i in.tree
```

Acknowledgment

ASTRAL code uses bytecode and some reverse engineered code from PhyloNet package (with permission from the authors).

Bug Reports:

contact: `astral-users@googlegroups.com`