StrAuto

Automation and Parallelization of STRUCTURE Analysis

User Manual Version 1.0

http://strauto.popgen.org

Vikram E. Chhatre[1,2] & Kevin J. Emerson[3]

[1]Dept. of Plant Biology, University of Vermont
[2]INBRE Bioinformatics Core, University of Wyoming
[3]Dept. of Biology, St. Mary's College of Maryland

crypticlineage@gmail.com
kjemerson@smcm.edu

February 18, 2018

Contents

1 For the Impatient

If you are a returning user and/or otherwise do not have time to read through the manual, just read the following quickstart instructions.

1. Make sure the following are installed on your computer/server:
   - Python 2.7.x or 3.0.x (see folder labeled `py3` for that version)
   - `STRUCTURE` 2.3.4 backend version (Pritchard et al., 2000)
   - `structureHARVESTER` (Earl and vonHoldt, 2012) (Optional)
   - GNU Parallel (Tange, 2011) (Optional)
2. Open the `input.py` file in a text editor (we recommend `vim`) and provide information about your data and the planned analysis.
3. An example data file `sim.str` based on Hubisz et al. (2009) has been provided for testing purposes.
4. Save the `input.py` file. Do not make any changes to the name of this file (`input.py`) including to the `.py` extension.
5. Launch the `StrAuto` program by typing `./strauto.1.py` in the terminal window.
6. StrAuto writes several output files: `runstructure`, `structureCommands`, `mainparams`, and `extraparams`. If needed, make any changes to `extraparams`, which only contains default options.
7. Launch Structure: `screen -S 'mysession' -d -m ./runstructure`. This launches a detached screen session.
8. You can reattach that session by typing `screen -r`
9. The session can be detached again with `[Ctrl-a + d]`, that is, type `[control a]` and `[d]` keys in quick succession.
10. Once Structure analysis is finished, a folder named `harvester` will contain a zip archive of the results.
11. The script will then collate the results using locally installed `structureHARVESTER`.

The rest of this document provides these instructions in more detail.

2 How to cite

If you found StrAuto useful in your research, please cite our article.

Chhatre, VE & Emerson KJ. StrAuto: Automation and parallelization of STRUCTURE analysis. BMC Bioinformatics (2017) 18:192. doi: http://dx.doi.org/10.1186/s12859-017-1593-0

3 Package Contents

Unpacking of the source archive will populate the current folder with following files. After unpacking, verify the MD5sum of the main script against the one provided in `strauto_1.md5`. Both scripts listed below work with Python version 2.7.x. For Python 3.0 functionality, see the `py3` folder.

- `strauto_1.py` - The main script
- `strauto_doc.pdf` - This guide
- `input.py` - Template file
- `sim.str` - Example data file from Hubisz et al (2009)
- `sampleStructureFile.py` - A program for subsetting your data for testing purposes.

4 Requirements

- Python 2.7.2 or 3.0.x (see folder named `py3`)
- Structure version 2.3.4 (http://goo.gl/NPjeb2) (Pritchard et al., 2000)
- Structure Harvester (Earl and vonHoldt, 2012) (https://users.soe.ucsc.edu/~dearl) (Optional)
- GNU Parallel (Tange, 2011) (http://www.gnu.org/software/parallel)
  This is an optional install. If you are working with very small data sets or do not have access to at least two processor cores, you can skip installing GNU Parallel. On Mac OS, this package is available through Fink and Homebrew. With Homebrew, you can install it by typing `brew install parallel` in the terminal. Verify the installation with `which parallel`.

5 What StrAuto Does

StrAuto and the resulting scripts will work on Mac OS and Linux. On Windows, same functionality can be obtained and on Windows via Cygwin. But the resulting shell script will only work with Mac OS and Linux. The StrAuto script:

1. Collects information about your data anlysis project from the template (`input.py`)
2. Checks whether Structure, structureHarvester and GNU Parallel are in system path
3. Writes Structure parameter files: `mainparams` and `extraparams`
4. Prepares a shell script (`runstructure`) and a companion commands file (`structureCommands`) when parallel implementation is sought.
5. When executed, the `runstructure` script will run Structure analysis, and store results and log files to respective folders.
6. A zip archive for Structure Harvester input is created.
7. If Harvester is available locally, it is run on the output files from above.

5.1 The `input.py` Template

This file is provided with the program as a template for entering information about your data analysis project. It contains 24 variables that need to be defined. The template refers to example data file `sim.str` based on Hubisz et al. (2009). Last four questions in the template concern parallelization and processing the results through Structure Harvester. If you would like to disable these options, make appropriate changes to the template.

5.2 Example Data File `sim.str`

We highly recommend that you set the burn-in and MCMC to 100 each and set up an example run with this data file. Once you verify that the trial run worked as intended, proceed with your own data.

5.3 Running StrAuto

Once you have the input.py template filled out completely, execute StrAuto as follows.

```
$ ./strauto_1.py
```

If you get an error message, your `$PATH` variable is not set correctly to execute the python binary. In that case, do the following (replace `/path/to/` with the actual path to the Python binary).

```
$ /path/to/python strauto_1.py
```

The program will present an introductory screen and wait for you to proceed. A carriage return allows the program to read the template `input.py` file. If the reading is successful, all parameter values will be displayed on screen for verification. If you find any errors, type (`q`)uit to exit the program and make corrections to your input file before proceeding. Once verified, type (`a`)ccept to begin writing the output files.

5.4 StrAuto Output Files

A successful run of StrAuto will generate following files:

```
$ ls -lh
-rw-r--r--@  1 user  group   1.0K Apr  2 11:42 extraparams
-rw-r--r--@  1 user  group   2.8K Apr  2 08:59 input.py
-rw-r--r--@  1 user  group   567B Apr  2 11:42 mainparams
-rwxr-xr-x@  1 user  group   634B Apr  2 11:42 runstructure
-rw-r--r--@  1 user  group   6.6K Apr  2 11:42 structureCommands
```

The program also checks the system `$PATH` and current directory for `STRUCTURE`, `structureHarvester` and `GNU Parallel` binaries. If one or more are not found, a message is printed to the screen. Output files are still generated, if these binaries are not found, but then we assume that you will populate the current directory with them.

6 Begin Structure analysis

Now you have everything you need to begin your structure run. If you ran StrAuto on your local workstation, but want to perform the analysis on a remote server, copy this folder in its entirety to that server. Then proceed with the analysis as follows.

```
$ screen -S my_session -d -m ./runstructure
```

Where:

- `screen` is a GNU (`http://gnu.org`) utility to run processes in the background
- `-S` switch names the screen session using assigned variable e.g. `my_session`
- `-d` switch creates a detached session
- `-m` invokes a new session (which is a detached session due to the `-d` switch we used)
- `./runstructure` executes the shell script

You will notice that the shell prompt has returned and is not showing any activity. This is because the Structure analysis is running in the background in a detached `screen` session. In order to see the progress, you will need to re-attach the session. Here is how to resume (notice the `-r` switch below) a session:

```
$ screen -r
```

To disconnect from the session (i.e. to detach it):

```
$ Ctrl-a + d
```

If you are wondering why we used `screen`, it is because `screen` allows the analysis to run in the background, as opposed to in an open-ended terminal session which, if you accidentally close or disconnect from the server, will disrupt your analysis. Structure analysis may, depending upon the size of your data and the number of markers used, be computationally intensive and will likely need hours, if not days to complete. The screen session can be detached and reattached any number of times without interrupting the process to monitor the progress. Even after the analysis is complete, the session remains in place so you could log back in to check progress. For more information, type `man screen` in the terminal window to read the user manual for this neat Unix utility.

Another Unix utility to check the status of the run is `top`. Try it as follows:

```
$ top -c
```

6.1 Output Files

If all went well, you should now have results. Check the following against your output. We are showing only relevant files/folders here for simplicity. Folder `results_f` contains individual results files for K clusters tested and the companion folder, `log` contains runtime screen output. Results from `structureHarvester` such as ΔK (Evanno et al., 2005) and relevant plots are are stored in the folder `harvester`. If you chose not to run harvester locally, only the zip archive ready for upload to its website will be stored in this folder.

```
$ ls -lh
drwxr-xr-x   3 user  group   102B Apr  2 12:44 harvester
drwxr-xr-x   6 user  group   204B Apr  2 12:44 log
drwxr-xr-x   6 user  group   204B Apr  2 12:44 results_f
-rw-r--r--   1 user  group    88B Apr  2 12:44 seed.txt
```

7 Behind the Scenes: `runstructure` script

If you plan on using this program, it is probably a good idea to familiarize yourself with how the `runstructure` script works, so that you can spot any errors or if the script is not doing what it should. The script follows these steps:

1. Create folders: `results_f`, `log`, and `harvester`
2. Create subfolders `K1` through `Kn` under `results_f` and `log` folders
3. Read `structureCommands` and execute `STRUCTURE` iteratively for all K clusters
4. Store the `results_f` and the `.log` files for each run in their respective folders
5. Once all runs are finished, move results folders `K1` through `Kn` inside `results_f`
6. Cull individual results files in one place and compress them into a zip archive
7. Move the zip archive to the `harvester` folder
8. If `structureHarvester` option was set to `1`, run the script on results and store the output in `harvester` folder. StrAuto expects that the harvester script is named `structureHarvester.py` and that it is available in the current directory.
9. Operation is complete.

7.1 A Note on Random Seed Number

To ensure assignment of unique seeds for the random number generator in each instance of structure, by default, we disable the RANDOMIZE function and initiate every new instance of structure using a unique random seed in the command line. This assures that replicate runs of structure are independent and that replicate runs of the same runstructure scripts are exactly the same.

8 Using High Performance Computing Cluster

This section describes protocol for deploying StrAuto based STRUCTURE analysis on high performance computing clusters (HPCC) to make use of the multi-node environment. The HPCCs almost always run on some flavor of the Linux OS. The jobs are queued using a scheduler. The examples below are specific to the **SLURM Workload Manager**. If your cluster uses **PBS**, your system administrator can provide further help.

8.1 Caution

Please consult your HPCC system administrator before launching these scripts. Engaging large number of nodes at once can overload the system. You have been warned.

8.2 Getting Started

Following example assumes that you have access to 3 compute nodes on a HPC cluster, each with 16 cores. We will also assume that you have run StrAuto and generated these files: `runstructure`, `structureCommands`, `mainparams`, and `extraparams`. The first two files (scripts) will need to be modified.

8.2.1 The `structureCommands` Script

Currently, your script looks something like this

```
./structure -K 1 -m mainparams -o k1/sim_k1_run1 2>&1 | tee log/k1/sim_k1_run1.log
./structure -K 1 -m mainparams -o k1/sim_k1_run2 2>&1 | tee log/k1/sim_k1_run2.log
./structure -K 1 -m mainparams -o k1/sim_k1_run3 2>&1 | tee log/k1/sim_k1_run3.log
./structure -K 1 -m mainparams -o k1/sim_k1_run4 2>&1 | tee log/k1/sim_k1_run4.log
```

Every line of this code needs to be prefixed with a `srun` statement which ensures one analysis process per physical core. This avoids hyperthreading and helps maximize performance scaling. The file should look like this after change (lines truncated due to space limitation).

```
srun -N1 -n1 --exclusive structure -K 1 -m mainparams -o...
srun -N1 -n1 --exclusive structure -K 1 -m mainparams -o...
srun -N1 -n1 --exclusive structure -K 1 -m mainparams -o...
srun -N1 -n1 --exclusive structure -K 1 -m mainparams -o...
```

In `vim` text editor, you can accomplish this with two commands. First command inserts the `srun` argument on each line. Second command is removing reference to local binary of structure (since we are using it as a module).

```
:%s/^/srun \-N1 \-n1 \-\-exclusive /g
:%s/\.\/structure/structure/g
```

8.2.2 The `runstructure` Script

We assume that 3 nodes (16x3=48 cores) will be deployed.

```
#!/bin/bash
#SBATCH --nodes=3
#SBATCH --tasks-per-node=16
#SBATCH --account=MY_ACCOUNT_NAME
#SBATCH --time=00:05:00

module load intel/16.3
module load structure/2.3.4
module load perl
module load parallel

mkdir results_f log harvester
mkdir k1
mkdir k2
mkdir k3
mkdir k4

mkdir log/k1
mkdir log/k2
mkdir log/k3
mkdir log/k4

cat structureCommands | parallel -j $SLURM_NTASKS --joblog parallel.log "{}"

... rest of the script follows
```

8.2.3 Submit the job

Once both files are edited as above and you have consulted with the sysadmins and ensured that it is okay to use whatever number of nodes you want, go ahead and submit the job. You can monitor the current status of the job at any time by using the `squeue` function.

```
$ sbatch runstructure
$ squeue -u username
```

9 Exploratory Analysis using Random Subsets

In order to do experimental tests in a reasonable amount of time with a large dataset, we recommend trial runs with a representative sample of your datatset. Our usage suggests that qualitative results details of the MCMC process are similar with datasets ranging from 500–20,000 loci. Therefore, to determine the length of MCMC chain for structure and other parameters, or to test different underlying models, we recommend that tests are done with ≈500 loci sampled from larger datasets. Once the user develops some intuition about the dataset and feels comfortable with the parameters to be used, they may then set up the process with the complete dataset.

To this end, we supply a python script `sampleStructureFile.py` that randomly samples a given number of loci from a structure-formatted input file.

9.1 Input File

Input for this program is a STRUCTURE formatted file. Currently the script only supports files where data for each individual is on two rows (i.e. `ONEROWPERIND=0`). If your data is formatted to indicate `ONEROWPERIND=1`, you will need to switch the format before using this script. Additionally it is assumed that (1) each locus column has a label (`MARKERNAMES=1`) and (2) all fields in the data file are tab separated (space separated genotype data will not work).

9.2 Usage

There are four required arguments for the script:

`-f` the number of fixed columns in the structure file (i.e. the number of columns that are not genotype data – these are assumed to be the left-most columns such as individual labels and location identifiers).

`-n` the number of loci that you would like to sample from the dataset

`-i` input file name

`-o` output file name

9.3 Output File

The output file is a structure-formatted file that contains the same fixed columns as the input file, and an additional n columns of genotype data (as determined by the `-n` parameter above). Additionally, a file called `sampleStructure.log` is created that lists the names of all of the loci that are retained in the sampled datafile.

9.4 Example

As an example, consider a structure-formatted file test.str that has one column with individual names and one with population names and 13,000 columns of genotype data. From this we wish to sample 700 loci. We would run

```
python sampleStructureFile.py -i test.str -o sampled_test.str -f 2 -n 700
```

This would generate two files – `sampled_test.str` that contained a column of individual names, a column of population names, and 700 columns of genotype data, and `sampleStructure.log` that gives information about the sampling process, including the names of all of the loci sampled.

10 Bibliography

D.A. Earl and B.M. vonHoldt. Structure harvester: A website and program for visualizing structure output and implementing the Evanno method. *Conservation Genetics Resources*, 4(2):359–361, 2012.

G. Evanno, S. Regnaut, and J. Goudet. Detecting the number of clusters of individuals using the software structure: a simulation study. *Molecular Ecology*, 14:2611–2620, 2005.

M.J. Hubisz, D. Falush, M. Stephens, and J.K. Pritchard. Inferring weak population structure with the assistance of sample group information. *Molecular Ecology Resources*, 9(5):1322–1332, 2009.

J.K. Pritchard, M. Stephens, and P. Donnelly. Inference of population structure using multilocus genotype data. *Genetics*, 155:945–959, 2000.

O. Tange. Gnu parallel - the command-line power tool. *The USENIX Magazine*, 36(1):42–47, Feb 2011. URL http://www.gnu.org/s/parallel.

11 Author Contributions & Acknowledgements

Original StrAuto was developed by VEC during his graduate training at Texas A&M University. It benefited from valuable feedback from Tomasz Koralewski and Tiffani Williams of Texas A&M University. VEC acknowledges indirect financial support from Texas A&M University's Genetics Graduate Program, Texas Forest Service and University of Vermont. KJE added parallelization, harvester analysis functionality and the random subset script. KJE acknowledges funding from St. Mary's College of Maryland. We thank Jared Baker of University of Wyoming Advanced Research Computing Center for help with testing StrAuto on HPC platform.