

# pong:

## fast analysis and visualization of latent clusters in population genetic data

**Developers (2016):** Katherine Z. Liu<sup>1,2</sup>, Tyler D. Devlin<sup>2,3</sup>, Aaron A. Behr<sup>\*</sup>, Sohini Ramachandran<sup>2,4\*</sup>

**Past developers:** Gracie Liu-Fang

<sup>1</sup>Department of Computer Science, Brown University, Providence, RI, USA

<sup>2</sup>Center for Computational Molecular Biology, Brown University, Providence, RI, USA

<sup>3</sup>Division of Applied Mathematics, Brown University, Providence, RI, USA

<sup>4</sup>Department of Ecology and Evolutionary Biology, Brown University, Providence, RI, USA

pong is freely available at <https://github.com/abehr/pong/> and can be installed using the Python package management system pip. This manual is the companion to version 1.4.6 of the software, released on 12/14/2016.

For a short guide to using pong, download the pong README at  
[http://brown.edu/Research/Ramachandran\\_Lab/projects/](http://brown.edu/Research/Ramachandran_Lab/projects/).

To request features or report issues when using pong, please visit  
<https://github.com/abehr/pong/issues>.

---

<sup>\*</sup>To whom correspondence should be addressed:

[aaron\\_behr@alumni.brown.edu](mailto:aaron_behr@alumni.brown.edu), [sramachandran@brown.edu](mailto:sramachandran@brown.edu).

# Software manual contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Citation information . . . . .	4
1.2	Overview of <code>pong</code> 's implementation . . . . .	4
<b>2</b>	<b>New features in <code>pong</code> version 1.4.6</b>	<b>5</b>
<b>3</b>	<b>Installing <code>pong</code></b>	<b>5</b>
3.1	Linux-specific tips . . . . .	5
3.2	Mac-specific tips . . . . .	6
<b>4</b>	<b>Running <code>pong</code> with example data</b>	<b>6</b>
<b>5</b>	<b>Input to <code>pong</code></b>	<b>7</b>
5.1	Clustering output (required) . . . . .	7
5.2	<i>filemap</i> file (required) . . . . .	8
5.3	Population labels (optional, but recommended) . . . . .	9
5.4	Population order and detailed names (optional) . . . . .	10
5.5	Custom colors for visualization (optional) . . . . .	11
<b>6</b>	<b>Additional input options</b>	<b>11</b>
6.1	Specifying an alternate distance metric . . . . .	11
6.1.1	Jaccard similarity (default) . . . . .	11
6.1.2	The sum of squared differences . . . . .	12
6.1.3	Average Manhattan distance . . . . .	12
6.1.4	$G'$ . . . . .	12
6.1.5	Custom similarity metrics . . . . .	12
6.2	Varying the threshold for determining modes . . . . .	13
6.3	Using a greedy approach to identify modes . . . . .	13
6.4	Server options . . . . .	13
6.4.1	Disabling the server . . . . .	13
6.4.2	Specifying the Tornado server's port . . . . .	14

<b>7</b>	<b>pong's output</b>	<b>14</b>
7.1	Output options . . . . .	14
7.1.1	Specifying the output directory . . . . .	14
7.1.2	Forced overwriting of an existing output directory . . . . .	14
7.2	Output files . . . . .	14
7.2.1	<i>params_used.txt</i> . . . . .	15
7.2.2	<i>result_summary.txt</i> . . . . .	15
7.3	Additional output ( <b>-v</b> flag) . . . . .	15
7.3.1	<i>best_alignment_across_K.txt</i> . . . . .	15
7.3.2	<i>best_alignment_per_K.txt</i> . . . . .	15
7.3.3	<i>cluster_matching_results</i> directory . . . . .	16
7.3.4	<i>distruct_perm_files</i> directory . . . . .	16
7.3.5	<i>runs</i> directory . . . . .	16
<b>8</b>	<b>pong's D3.js-based Visualization</b>	<b>16</b>
8.1	Main visualization . . . . .	16
8.2	Interactive features in pong's visualization . . . . .	17
8.2.1	Detecting resized browser windows . . . . .	17
8.2.2	Tooltip displaying population information . . . . .	17
8.2.3	Highlighting specific clusters The user can click on a single cluster within a barplot and highlight membership in that cluster across all values of $K$ (and all modes; see Section 8.2.6). The selected cluster will be displayed in full color across all barplots, while other clusters will appear white. To undo the selection, click anywhere within a barplot.	17
8.2.4	Highlighting specific populations . . . . .	17
8.2.5	Zooming into barplots . . . . .	17
8.2.6	Visualizing multimodality for a given value of $K$ . . . . .	18
8.3	Downloading barplots . . . . .	18
<b>9</b>	<b>Examples</b>	<b>19</b>
9.1	1000 Genomes phase 3 dataset, main visualization . . . . .	19
9.2	Chicken dataset from Rosenberg et al. [2001], visualizing modes . . . . .	20

# 1 Introduction

A series of generative models known as “mixed-membership models” have been developed that model grouped data, where each group is characterized by a mixture of latent components. One well-known mixed-membership model is latent Dirichlet allocation [Blei et al., 2003], in which documents are modeled as a mixture of latent topics. Another widely used example is the model implemented in the population-genetic program STRUCTURE [Pritchard et al., 2000, Falush et al., 2003, Hubisz et al., 2009, Raj et al., 2014], where individuals are assigned to a mixture of latent *clusters*, or populations, based on multilocus genotype data; this same model underlies the program ADMIXTURE [Alexander et al., 2009].

In the population-genetic application of mixed-membership models, for a given multilocus genotype dataset with  $N$  individuals and a value for  $K$  (the number of populations, or *clusters*, from which the individuals derive ancestry), the output produced by STRUCTURE and ADMIXTURE is an  $N \times K$  matrix of *membership coefficients*, referred to here as a “ $Q$  matrix”. Each entry  $q_{ij}$  represents the membership coefficient for individual  $i$  in cluster  $j$ , and can be interpreted as the inferred proportion of individual  $i$ ’s alleles inherited from the  $j$ th cluster.

A common pipeline when applying model-based clustering methods to genotype data is to increment  $K$  from 2 to some user-chosen maximum value, and to conduct many runs of clustering inference for a fixed value of  $K$ , resulting in the routine production of around 100  $Q$  matrices for the analysis of a single dataset. Post-hoc processing of these  $Q$  matrices can be laborious, due to changing values of  $K$  and because the stochastic nature of clustering methods produces different  $Q$  matrices for identical inputs across independent iterations of clustering inference.

Current algorithms designed to process and visualize  $Q$  matrices face three challenges

1. **label switching**: for the same inputs, column  $\vec{q}_{\cdot j}$  in the  $Q$  matrix produced by one run may not correspond to column  $\vec{q}_{\cdot j}$  in the  $Q$  matrix produced by another run [Stephens, 2000, Jasra et al., 2005, Jakobsson and Rosenberg, 2007].
2. **multimodality**: where, even after adjusting for label switching, different  $Q$  matrices can be inferred from the same input data (see also CLUMPP by Jakobsson and Rosenberg [2007]).
3. **alignment-across- $K$** : when the input parameter  $K$  is changed (all other inputs being equal), there is no column-permutation of an  $Q_{N \times K}$  matrix that exactly corresponds to any  $Q_{N \times (K+1)}$  matrix (see also CLUMPAK by Kopelman et al. [2015]).

Here, we present **pong**, a package consisting of a new algorithm for fast post-hoc analysis of clustering inference output from population genetic data and a custom, interactive D3.js-based data visualization (D3.js; <https://github.com/mbostock/d3>). Our package solves the challenges of accounting for label switching, characterizing multiple modes, and aligning  $Q$  matrices across values of  $K$  by constructing weighted bipartite graphs for each pair of  $Q$  matrices depicting similarity in membership coefficients between clusters. **pong** allows

users to begin interacting with their results potentially seconds after clustering inference has been completed. When applied to clustering inference output, `pong` displays an accurate representative  $Q$  matrix for each mode for each value of  $K$ , clarifies differences among modes that are difficult to identify through visual inspection, and characterizes results that may be overlooked when interpreting output by hand.

This manual provides comprehensive information on customizing `pong`'s algorithm and visualization. A quick overview for installing and using `pong` can be found in the `pong` reference card, available at [http://brown.edu/Research/Ramachandran\\_Lab/projects/](http://brown.edu/Research/Ramachandran_Lab/projects/).

## 1.1 Citation information

For more details and comparisons of `pong` to existing methods for visualization of clustering inference from population genetic data, please refer to our paper by Behr *et al.*, doi: <http://dx.doi.org/10.1101/031815>, available at biorxiv (<http://biorxiv.org/content/early/2015/11/14/031815>).

## 1.2 Overview of `pong`'s implementation

`pong`'s back end algorithm is written in Python and `pong` is run with a one-line command in the terminal. While providing covariates is strongly advised so visualizations can be annotated with relevant metadata, `pong` only requires one tab-delimited file containing relative paths to each  $Q$  matrix (detailed further in Section 5.2). The command line can also contain a series of flags which allows the user to customize the visualization. Executing a command in `pong` opens a Tornado web server (<http://www.tornadoweb.org/>), and eventually prompts the user to open a web browser and connect to a specified localhost. This action then renders the visualization developed by `pong`'s front end.

After alignment within and across  $K$ , `pong`'s back end generates a JavaScript Object Notation (JSON) file for data interchange with the front end. The JSON entries vary but, at a minimum, contain paths to the input  $Q$  matrices, user-created run identification codes (or *runIDs*), information on identified modes, and the color permutation used for visualization. To render its visualization, `pong` does not preload  $Q$  matrices into the JSON sent to the front end. Instead, the user's actions in the browser window lead to requests for data, such as  $Q$  matrices, from the locally run server; these data are sent to `pong`'s front end via sockets. This increases the efficiency of the visualization.

`pong`'s visualization is implemented in Javascript and uses D3.js, a library that allows developers to create dynamic and interactive web applications with bound data. This visualization replaces DISTRUCT [Rosenberg, 2004] and other methods for the graphic display of clustering inference output, and produces a visualization that is user-friendly, visually appealing and informative regarding the input sample's population structure. If the user does not provide a color permutation to `pong`, the colors used to generate the visualization are chosen from color groupings available in ColorBrewer (<http://colorbrewer2.org/>).

What sets `pong` apart from existing methods for the graphical display of population structure is a series of interactive features. In the browser’s main visualization, the user may click on any population — or set of populations, by holding the SHIFT key down – to highlight the selected group’s genome-wide ancestry across values of  $K$ . Clusters can be highlighted by clicking on a particular cluster color, and users can zoom into barplots. When mousing over a population, color swatches corresponding to the average membership (as a percentage) in each cluster for that population are displayed in a tooltip. Within each dialog box characterizing modes for a given value of  $K$ , selecting a checkbox on the top right allows the user to highlight differences between the major mode’s representative plot (the top plot in the dialog box) and each minor mode’s representative plot. Mousing over populations within the dialog box also displays population-level average membership in each cluster to the right of the dialog box.

## 2 New features in `pong` version 1.4.6

The following features have been added to `pong` with this version:

- Improved documentation and error handling around valid/invalid characters in filemaps (Section 5.2).
- Unicode escape characters in `--col_delim` are now properly handled (Section 5.1).
- Checking whether the user has the Python package `NetworkX` ( $\geq 1.10$ ). When installing `pong` with `pip` (Section 3), we now check for `NetworkX` and install it if the user does not already have `NetworkX`.

## 3 Installing `pong`

`pong` has been tested on Mac OS X (10.8-10.10), Linux (Ubuntu 15.04, Linux Mint 17.2), and Windows 7. `pong` is hosted on PyPI (<https://pypi.python.org/pypi>) and can thus be easily installed using the Python package management system `pip`. Running `pong` requires Python 2 (version 2.7.8 or later) and a modern web browser; `pong` has been tested on Chrome, Firefox, and Safari. `pong` is not compatible with Internet Explorer. The source code for `pong` is available at <https://github.com/abehr/pong>.

To install `pong`, run:

```
pip install pong
```

### 3.1 Linux-specific tips

On some Linux systems, installation of `pong` may fail due to a permissions error. In that case, try running

```
sudo pip install pong
```

You will be prompted to enter in your computer login password. If that doesn't work either, try running the command as the super user.

```
su # you will be prompted to enter an administrator password
pip install pong
exit # to quit out of super user mode
```

## 3.2 Mac-specific tips

Note that the Apple system default Python cannot run pong. Python 2.7 and pip can be installed manually, but we find that setup is easiest if the user has both [Homebrew](http://brew.sh) (<http://brew.sh>) and [Homebrew-installed Python](#). Using Homebrew, pong's dependencies, and pong itself, can be installed / updated as follows:

```
brew install python / brew upgrade python
pip install pip / pip install --upgrade pip
pip install pong / pip install --upgrade pong
```

We will continue to add features to pong and will post regarding new releases on pong's git repository (<https://github.com/abehr/pong/>). To upgrade, users should run

```
pip install --upgrade pong
```

## 4 Running pong with example data

pong is executed through the command line in a terminal window. To see a complete list of options with descriptions, run:

```
pong -h
```

To run pong from the command line as in the above example, make sure the filepath to python executables on your computer is in your PATH. If this is not the case, then you can run pong commands using the prefix

```
python path_to_python_executables/pong [...continue command line...]
```

Users can download an example dataset from [the Ramachandran Lab website](#) or from [the Ramachandran Lab Data Repository](#). This data is comprised of ADMIXTURE [Alexander et al., 2009] runs on the 1000 Genomes dataset phase 3 (final variant set released on November 6, 2014, with first- and second- degree relatives removed; see also Consortium [2015]). Our input to ADMIXTURE was 225,705 genome-wide single-nucleotide variant genotypes from 2,426 unrelated individuals. In the example dataset, ADMIXTURE was run with  $K$  ranging from 2 to 8 and 8 runs per value of  $K$ ; this produced a total of 56  $Q$  matrices.

To analyze the example dataset provided with `pong`, navigate into the unzipped example dataset directory `pong-example-data_1kG-p3` and run:

```
pong -m pong_filemap -n pop_order_expandednames.txt -i ind2pop.txt
```

Information regarding `pong`'s application to the input data will be displayed to your terminal window. Once the back end has completed running, `pong` initializes a local server and informs the user of its location. A message will be displayed in the terminal window telling the user to navigate to `http://localhost:4000` (port 4000 is used by default; the user can change the port on which `pong` operates with the command line option `--port`). Once the user navigates to `http://localhost:4000`, `pong` will detect a new browser connection and begin loading the its visualization of the  $Q$  matrices. `pong` also has a website tour, implemented using <http://bootstraptour.com/>, to introduce users to features in `pong`'s visualization.

## 5 Input to `pong`

This section details the required and optional inputs that `pong` accepts.

### 5.1 Clustering output (required)

`pong` accepts clustering output files, i.e.  $Q$  matrices, from a variety of clustering inference methods. Note that users must remove leading/trailing rows from input  $Q$  matrices (i.e. for a dataset with  $N$  samples, every  $Q$  matrix file in the *filemap* should have exactly  $N$  lines). `pong` does not accept  $Q$  matrices with  $K = 1$ , which do not provide any information.

Two command line flags can be used to customize `pong`'s treatment of these  $Q$  matrices; these flags are detailed below.

#### 1. Users can indicate that `pong` should ignore columns in $Q$ matrices.

Command line flag: `-c number / --ignore_cols number`

If `-c` is not specified, 0 is used by default. Because some clustering inference methods report individual covariates as leading columns before along individual membership coefficients, the user can use this flag to indicate that `pong` should skip a certain number



of leading columns in each row of the input  $Q$  matrices before parsing individual membership coefficients.

For example, use `-c 5` for some versions of STRUCTURE output, because the first five columns contain individual-level metadata.

Trailing columns of covariates in a  $Q$  matrix file will not be parsed by `pong`. That is, after `-c number` leading columns are ignored, only the first  $K$  columns are used by `pong`, where  $K$  is specified by the user in the *filemap*).

## 2. Users can provide $Q$ matrices with any column delimiter.

Command line flag: `--col_delim`

By default, `pong` parses  $Q$  matrix files as whitespace-delimited. However, the user provide a specific column delimiter in quotes — such as `,` (comma) or `\t` (tab) — to be used for parsing all  $Q$  matrices with this flag.

## 5.2 *filemap* file (required)

Command line flags: `-m filename / --filemap filename`

A small amount of information about the input  $Q$  matrices **must** be provided to `pong`, in the form of a *filemap*. This is the only command-line flag required to run `pong`. The *filemap* must be a three-column, tab-delimited file. Each  $Q$  matrix is represented by a three-column line in the *filemap* with the following format:

**Column 1.** The runID, a unique label for the  $Q$  matrix (e.g. the string “run5\_K7”). Note: A runID must begin with a letter (A-Z/a-z), followed by any number of hyphens (-), underscores (\_), letters, or numbers. Other characters are not allowed in runIDs. Hashmarks (#) can be used in the *filemap* to indicate the start of a comment.

**Column 2.** The  $K$  value for the  $Q$  matrix. Each value of  $K$  between  $K_{min}$  and  $K_{max}$  must be represented by at least one  $Q$  matrix in the *filemap*; if not, `pong` will abort.

**Column 3.** The path to the  $Q$  matrix, relative to the location of the *filemap*. Thus, if the *filemap* is in the same directory as the  $Q$  matrix file, this is just the name of the  $Q$  matrix file. Note that the metadata provided in the *filemap* allow the user to apply `pong` to  $Q$  matrices in multiple directories in the user’s computer. The path cannot contain a hashmark (#) because it will be interpreted as a comment.

It is important that the columns of the *filemap* are in the right order and that the file is tab-delimited. For reference, Table 1 below shows a portion of the *filemap* corresponding to the example dataset.

Table 1: Example lines of a tab-delimited *filemap* with a unique runID, a K value, and a filepath for every iteration in the 1000 Genomes example dataset. An example comment is shown in line 9.

	runID	K value	filepath
1	k2r1	2	run1/pruned_filtered_1kg_phase3.2.Q
2	k2r2	2	run2/pruned_filtered_1kg_phase3.2.Q
3	k2r3	2	run3/pruned_filtered_1kg_phase3.2.Q
4	k2r4	2	run4/pruned_filtered_1kg_phase3.2.Q
:	:	:	:
:	:	:	:
9	#another test comment		
10	k3r1	3	run1/pruned_filtered_1kg_phase3.3.Q
:	:	:	:
:	:	:	:

### 5.3 Population labels (optional, but recommended)

Command line flags: `-i ind2pop_arg / --ind2pop ind2pop_arg`

In most population-genetic datasets, individuals have a population label (or code, or number), that indicate that individual’s origin; we refer to the set of these labels as *ind2pop* data. If provided with this information, *pong*’s visualization will group individuals into populations by a label, partition populations with black lines, and sort individuals within each population by their membership in the population’s major cluster (the cluster in which the population has the greatest membership, when membership coefficients are summed over all individuals). *pong* performs this sorting operation based on cluster membership for each population on the bottom-most plot in the main visualization, which is the representative run of the major mode of the highest *K* value across all input *Q* matrices. *pong* then propagates the order of individuals determined after sorting through all the other visualized *Q* matrices, keeping the order of individuals consistent across the visualization.

The argument to the *ind2pop* command line flag can be either of the following:

- An integer, representing the *Q* matrix column number that contains *ind2pop* data. For example, use `-i 4` for most versions of STRUCTURE output.
- The path to an *ind2pop* file, where line *i* of this file contains the *ind2pop* data (i.e. population label/code/number) for the individual represented by line *i* of the *Q* matrix files. This file should contain a single column, with the same number of rows as individuals in the input *Q* matrices (Table 2). Thus, blank lines are not tolerated.

Table 2: Example lines from the *ind2pop* file for the 1000 Genomes example dataset. In the single column file, each line corresponds to an individual’s population assignment from the 1000 Genomes data (2426 individuals).

GBR
GBR
GBR
GBR
:
:
ITU

Table 3: Example lines from a *pop\_order* file. The first column contains population labels from the *ind2pop* file; the top-to-bottom order of these labels will be the left-to-right order in which the population are displayed in *pong*’s visualization. The second column contains expanded, more descriptive population names which will be displayed in *pong*’s visualization instead of the three-letter codes in the first column. The information in the second column is optional.

YRI	Yoruba in Ibadan, Nigeria
LWK	Luhya in Webuye, Kenya
:	:
:	:
GBR	British in England and Scotland
IBS	Iberian Population in Spain
GIH	Gujarati Indian from Houston, Texas

## 5.4 Population order and detailed names (optional)

Command line flags: `-n filename / --pop_names filename`

Besides *ind2pop* data, the user may provide an additional file specifying the left-to-right order in which *pong* should display the populations using either of the command-line options above. This file should have one population label/code/number per line that matches labels/codes/numbers used in *ind2pop* data (see first column in Table 3). The top-to-bottom order of population labels in this file will correspond with the left-to-right order of the populations in *pong*’s visualization. *pong* will abort execution if the user provides population order and labels information but not *ind2pop* data.

Suppose the input *ind2pop* data are numbers or three-letter codes. In such cases, the visualization may benefit from more descriptive population labels, and the user can add a second column (tab-delimited) to the population order file containing population names (see second column in Table 3). Space characters *are allowed* in these names. Use `pop_order.txt` and `pop_order_expandednames.txt` in the example dataset for reference.

Note that users can group individuals by any covariate in *ind2pop* data, such as language spoken or geographic region.

## 5.5 Custom colors for visualization (optional)

Command line flags: `-l filename / --color_list filename`

`pong` has a set of default colors used to denote membership in clusters (unless the maximum  $K$  value across all input  $Q$  matrices,  $K_{\max}$ , is greater than 26). However, the user can provide `pong` with a file containing a set of custom colors to use for visualization using either of the above command-line options. This file must contain at least  $K_{\max}$  colors, with one color per line; if the user provides more colors than  $K_{\max}$  only the first  $K_{\max}$  colors will be used in the visualization.

Because `pong`'s visualization is web-based, colors can be provided in any format that is accepted by Cascading Style Sheets (CSS) (see <http://www.w3.org/Style/CSS/> for an overview). That is, the file with colors can contain hexadecimal color codes (e.g., `#ff0000`), RGB values (e.g., `rgb(255,0,0)`), RGBA values (e.g., `rgba(255,0,0, 0.4)` for red with 40% opacity) or HTML color strings (e.g., `red`). It is acceptable to mix these various color codes within a single custom color file.

Note that `pong`'s visualization uses the color white to highlight differences among modes given a fixed value of  $K$ . Users should therefore refrain from using white in any custom color file.

## 6 Additional input options

### 6.1 Specifying an alternate distance metric

Command line flag: `--dist_metric` [argument options are detailed in this section]

We have implemented several metrics to compute similarity between clusters in input  $Q$  matrices. We strongly recommend users use `pong`'s default metric, Jaccard similarity ( $\mathcal{J}$ ; Equation 1) because we find that  $\mathcal{J}$  is the most conservative metric with respect to grouping runs into modes given a fixed value of  $K$ . Several other options are available for interested investigators, detailed below.

The possible arguments to the command line flag `--dist_metric` are `jaccard` (this is the default; Equation 1), `sum_squared` (Equation 2), `percent` (Equation 3), and `G` (Equation 4).

#### 6.1.1 Jaccard similarity (default)

$\mathcal{J}$  has been derived from the Jaccard index used in set comparison. When using this metric, the similarity value between two clusters does not increase if individuals have no membership in either cluster. This pairwise similarity metric  $\mathcal{J}$  is computed as follows: For a given pair

of clusters  $\{\vec{q}_{\bullet a}, \vec{q}_{\bullet b}\}$ , let  $N^*$  be the set of indices for which at least one of  $\{\vec{q}_{\bullet a}, \vec{q}_{\bullet b}\}$  has a nonzero entry; that is,  $N^* = \{i \in [1, N] : q_{ia} + q_{ib} > 0\}$ . Then,

$$\mathcal{J}(\vec{q}_{\bullet a}, \vec{q}_{\bullet b}) = 1 - \sqrt{\frac{\sum_{i \in N^*} (q_{ia} - q_{ib})^2}{2|N^*|}} \quad (1)$$

### 6.1.2 The sum of squared differences

Command line flag prototype: `--dist_metric sum_squared`

`sum_squared` measures pairwise similarity by computing the sum of the squared differences between clusters. Given a pair of clusters  $\{\vec{q}_{\bullet a}, \vec{q}_{\bullet b}\}$ , and the total number of individuals  $N$ , the `sum_squared` distance metric is computed as follows:

$$\text{sum\_squared}(\vec{q}_{\bullet a}, \vec{q}_{\bullet b}) = 1 - \frac{\sum_{i \in N} (q_{ia} - q_{ib})^2}{N} \quad (2)$$

### 6.1.3 Average Manhattan distance

Command line flag prototype: `--dist_metric percent`

`percent` calculates the absolute percent similarity between a pair of clusters,  $\{\vec{q}_{\bullet a}, \vec{q}_{\bullet b}\}$ , with the following equation:

$$\text{percent}(\vec{q}_{\bullet a}, \vec{q}_{\bullet b}) = 1 - \frac{\sum_{i \in N} |(q_{ia} - q_{ib})|}{N} \quad (3)$$

### 6.1.4 $G'$

Command line flag prototype: `--dist_metric G`

The  $G'$  distance metric was defined by [Jakobsson and Rosenberg \[2007\]](#) as follows:

$$G'(\vec{q}_{\bullet a}, \vec{q}_{\bullet b}) = 1 - \sqrt{\frac{\sum_{i \in N} (q_{ia} - q_{ib})^2}{2|N|}} \quad (4)$$

### 6.1.5 Custom similarity metrics

pong's implementation is designed such that users familiar with Python and NumPy can add their own similarity metrics to the source code. Users interested in extending pong in this manner should contact [Aaron Behr](#).

## 6.2 Varying the threshold for determining modes

Command line flags: `-s value / --sim_threshold value`

`pong` relies `sim_threshold` (0.97 by default; can take on real values in  $[0,1]$ ) when characterizing multimodality for a fixed value of  $K$ . Given a value of  $K$ , `pong` uses algorithms for the assignment problem [Manber, 1989, Kuhn, 1955, 1956, Munkres, 1957] to align each pair of runs; here we use “align” to denote constructing a bipartite perfect matching between clusters among a pair of runs. `pong`’s objective is to find the maximum-weight alignment for each pairs of runs within a value of  $K$ . The average edge weight for each maximum-weight alignment is stored to construct a pairwise similarity graph among all runs at that value of  $K$ . In this pairwise similarity graph, each node is a run, and each edge denotes the average edge weight of the maximum-weight alignment between the pair of runs it connects. By default, an edge with weight less than `sim_threshold`= 0.97 is not added to the pairwise similarity graph before modes are identified. Note that setting `sim_threshold`= 1 will force each run to be in a separate mode; users may wish to set the threshold to 1 in order to visualize all the input  $Q$  matrices given to `pong`.

`pong` uses the package `NetworkX` [Hagberg et al., 2008] to find disjoint cliques within the pairwise similarity graph; these disjoint cliques are the modes.

## 6.3 Using a greedy approach to identify modes

Command line flag: `-g / --greedy`

For a given value of  $K$ , `pong` defines modes by identifying disjoint cliques in the graph of pairwise similarities between all pairs of runs. If a pair of runs has pairwise similarity less than `sim_threshold` (Section 6.2), the edge connecting that pair of runs is not added to the graph of pairwise similarities. This makes the graph sparse but does not ensure that cliques will be disjoint; that is, there may be some runs that occur in multiple modes when `pong` identifies cliques (using the package `NetworkX`; [Hagberg et al., 2008]).

If `pong` identifies cliques that are not disjoint, `pong` will prompt the user by default with a choice: whether to continue with the greedy algorithm, or to exit and re-run with different parameters (such as a higher `sim_threshold`). The greedy algorithm iteratively removes the maximum clique from the graph of pairwise similarities, and is thus guaranteed to produce disjoint cliques (modes). The command flag `-g` or `--greedy` will simply force `pong` to use this greedy approach, thereby freeing the user from waiting for `pong`’s prompt.

## 6.4 Server options

### 6.4.1 Disabling the server

Command line flag: `--disable_server`

By default, `pong` initializes a local Tornado web server instance on which to host the D3 interactive visualization. Users can disable `pong`’s server to run its algorithm without visualizing

results. The outputs that `pong` gives to the user beyond its visualization are detailed in Section 7.

### 6.4.2 Specifying the Tornado server's port

Command line flags: `-p number / --port number`

This command-line option allows a user to specify the port on which the server is locally hosted. The default value is 4000.

## 7 `pong`'s output

Besides its visualization, `pong` provides the user with multiple types of output regarding modes and alignment of  $Q$  matrices. In Section 7.1, we detail command-line options the user can use to customize `pong`'s output. In Section 7.2, we discuss output files produced by `pong`.

### 7.1 Output options

#### 7.1.1 Specifying the output directory

Command line flags: `-o directory_path / --output_dir directory_path`

By default, `pong` makes an output directory called `pong_output_datetime`, where *datetime* is the current system date and time. However, `pong` allows the user to specify the relative path to the output directory where output files will be written using this flag. If the directory doesn't exist, the directory will be created. If the directory does exist, `pong` gives the user the option to overwrite existing files or to abort.

#### 7.1.2 Forced overwriting of an existing output directory

Command line flags: `-f / --force`

This flag will automatically overwrite the output directory, if it already exists.

### 7.2 Output files

`pong` generates results from its characterization of modes and alignment procedure that are printed to a series of output files in the output directory (see Section 7.1.1). Two output files are automatically reported when executing `pong`:

### 7.2.1 *params\_used.txt*

The main parameters used to run `pong` are printed in the output file *params\_used.txt*. This includes the filepath for the *filemap*, the similarity metric computed between clusters, the similarity threshold used to determine multimodality, and the executed command line.

### 7.2.2 *result\_summary.txt*

This file details the modes identified by `pong`. For each value of  $K$ , *result\_summary.txt* contains the total number of iterations analyzed, runIDs for the representative runs of each mode, and the runIDs of all the iterations in the each mode. If there are multiple modes identified for a value of  $K$ , we report the **Avg sim between modes**, which indicates average pairwise similarity across pairs of representative runs for each mode at that value of  $K$ . (Note that “pairwise similarity” is defined as the average edge weight in the maximum-weight alignment between a pair of runs.) If there is more than one  $Q$  matrix in a mode, the value of **Avg sim within** represents the average pairwise similarity across all pairs of runs within that mode.

## 7.3 Additional output (-v flag)

Command line flags: `-v` / `--verbose`

Using `pong`’s verbose command-line flag will produce more detailed output files and additional output directories, which we describe in Sections 7.3.1-7.3.5.

### 7.3.1 *best\_alignment\_across\_K.txt*

This file details the permutation of clusters that produces the maximum-weight alignment across representative runs of major modes from  $K = k$  to  $K = k + 1$  ( $k = K_{\min}, \dots, K_{\max} - 1$ ). The user-provided runIDs (see the *filemap* in Section 5.2) of each major mode’s representative run are printed in this file.

### 7.3.2 *best\_alignment\_per\_K.txt*

This file details information from aligning  $Q$  matrices within a fixed  $K$  value. For each  $K$  value, this file lists the runID of each  $Q$  matrix and the permutation of the matrix columns that produces the maximum-weight alignment with the representative run of the major mode. Representative runs for each value of  $K$  are indicated by an asterisk (\*) on the right side of the permutation.



### 7.3.3 *cluster\_matching\_results* directory

Each file in this directory details the similarity metric value between each pair of clusters, as well as the average pairwise similarity, for (i) pairs of  $Q$  matrices within a value of  $K$  and (ii) pairs of  $Q$  matrices that are representative runs of major modes across consecutive values of  $K$ . Recall that the edge weights reported are determined by the `dist_metric` used (see Section 6.1).

### 7.3.4 *distruct\_perm\_files* directory

If the user provides `pong` with a custom color file (Section 5.5) then, in this directory, `pong` outputs a DISTRUCT [Rosenberg, 2004] color permutation file for each run, and annotates the filenames of each mode’s representative run. These files can be used as input into DISTRUCT [Rosenberg, 2004] to visualize  $Q$  matrices of particular runs (versus `pong`’s summary of modes, which uses a single representative  $Q$  matrix for each mode). For users familiar with the command line, using the `-v` flag and DISTRUCT while disabling `pong`’s server (Section 6.4.1) will enable the production of static graphics without viewing `pong`’s interactive visualization; these options thus allow `pong` to be integrated into automated pipelines.

### 7.3.5 *runs* directory

`pong` elects representative runs for each mode it detects from the input set of  $Q$  matrices. When the `-v` / `-verbose` flag is used, `pong` then copies all  $Q$  matrices designated as representative runs to the *runs* directory.

## 8 `pong`’s D3.js-based Visualization

`pong` produces a visualization implemented in Javascript using the D3.js library. It is unique in its interactive and dynamic graphical display of clustering inference output, providing more information than existing approaches in a visually appealing and user-friendly way. This section details the interactive aspects of the visualization.

### 8.1 Main visualization

Once `pong`’s algorithms have finished analyzing the dataset, the user will be prompted to open a browser and connect to a specified localhost. After this connection is established, `pong`’s back end will send information in JSON files to the front end to load a main visualization. This main visualization displays the representative runs for the major mode at each value of  $K$  as a barplot using Scalable Vector Graphics (SVG; see Figure 2). In our visualization, as in DISTRUCT [Rosenberg, 2004], each individual is plotted as a stacked vertical line of  $K$  colored line segments that represent the individual’s membership derived from each cluster. Alongside each barplot, we display the corresponding value of  $K$ , the number of runs

classified in the major mode at each value of  $K$ , and the average pairwise similarity across all pairs of runs in the major mode. If the user provided `pong` with *ind2pop* information (see Section 5.3), population labels are displayed following all representative runs of major modes and populations are separated by black lines in each barplot.

## 8.2 Interactive features in `pong`'s visualization

### 8.2.1 Detecting resized browser windows

The user may want to resize her browser window in order to view a larger visualization or to fit a smaller screen. `pong` detects resizing events in the browser window, and prompts the user to reload the browser in order to render the visualization to the browser's new dimensions.

### 8.2.2 Tooltip displaying population information

Mousing over a specific population in any barplot in `pong`'s visualization displays a tooltip which reports the population's label (or covariate; Section 5.3), the number of samples in the population, and the average membership in each cluster across all individuals in the population (for clusters where population-level membership is  $\geq 0.5\%$ ).

**8.2.3 Highlighting specific clusters** The user can click on a single cluster within a barplot and highlight membership in that cluster across all values of  $K$  (and all modes; see Section 8.2.6). The selected cluster will be displayed in full color across all barplots, while other clusters will appear white. To undo the selection, click anywhere within a barplot.

### 8.2.4 Highlighting specific populations

The user can click on any population label — or a set of populations by holding the SHIFT key — to highlight their membership coefficients across all values of  $K$  (and all modes; see Section 8.2.6). All selected populations are displayed in full color, while other populations appear in dark grey. To undo the selection, simply click on any population label (do not hold the SHIFT key).

### 8.2.5 Zooming into barplots

To zoom into a portion of a barplot, the user can mouse over the portion of interest and scroll on his mouse. Zooming is centered around the location of the user's cursor. Note that it is possible to zoom in such a way that a population at the edge of the barplot may be partially cut off. As a result, the population labels for populations at the edges of the barplot may no longer be visible at the bottom of the main visualization. Additionally, the population's tooltip (Section 8.2.2) may appear off to the side of the visualization, since tooltips are positioned at a population's center (wherever it may be).

### 8.2.6 Visualizing multimodality for a given value of $K$

For any value of  $K$  that has more than one distinct mode, a button to the right of the major mode barplot denotes the number of additional modes at that value of  $K$ . Clicking this button opens a dialog box, where the top barplot displayed is the representative run of the major mode at that value of  $K$ , followed by barplots representing each of the minor modes. Next to each barplot in the dialog box, we display information regarding how many runs were grouped into each mode and the runIDs of each mode’s representative run. Any interactive action in the main visualization (see Sections 8.2.2-8.2.5) will automatically propagate to the visualization in all dialog boxes.

The dialog box also has a checkbox on the top right corner that allows users to highlight multimodality. When this box is checked, clusters in the minor modes’ representative runs with similarity of at least `sim_threshold` (0.97 by default) to corresponding clusters in the major mode are redrawn as white, while clusters that are less similar maintain their full color to highlight the difference between each minor mode and the major mode. For some datasets and values of  $K$ , all corresponding clusters across modes will appear very similar while actually having similarity less than `sim_threshold`. In this case, the user is given appropriate messaging in the dialog, with the recommendation of lowering `sim_threshold` in `pong`’s command line to enable multimodality highlighting (Section 6.2). When multimodality highlighting is turned on, the ability to click to highlight clusters is temporarily suspended. Conversely, multimodality highlighting is disabled if a cluster has been clicked (indicated by a change in cursor when the user’s mouse approaches the checkbox).

The dialog header also reports the average pairwise similarity across all pairs of representative runs of modes. To close the dialog box, users can click the ‘x’ on the upper right side, or click outside the dialog box.

## 8.3 Downloading barplots

Every barplot displayed by `pong` can be downloaded in multiple formats for use in manuscripts and presentations: Portable Document Format (PDF; via print dialog), Portable Network Graphics (PNG; via download), and Scalable Vector Graphics (SVG; via download). If *ind2pop* data is given to `pong` (Section 5.3), population/covariate labels will also be displayed in all downloaded visualizations.

Barplots can be downloaded individually by clicking the printer icon or download icon buttons to the left of each barplot; this functionality exists in both the main visualization and in each dialog box. All barplots in the main visualization (or the set of barplots representing each mode in a dialog box) can be downloaded as a group using buttons at the bottom of the browser window (or dialog box). The download/print feature mimics what the user sees in the browser, so if a specific part of the barplots is highlighted (Sections 8.2.3-8.2.6), that particular state of the visualization will be downloaded/printed.

Upon clicking a print button, `pong` initiates a print dialog in the browser that opens a new tab where users can either print the barplot (with population/covariate labels, if those labels

exist) or choose the “Save to PDF” option to download the visualization of their input  $Q$  matrices; these downloaded PDFs are also editable in programs like [Adobe Illustrator](#). A potential drawback to `pong`’s use of print dialogs for downloading PDFs is that pagination of the visualization will occur based on the user’s print settings; this may cut off static visualizations for datasets with many individuals (see “Troubleshooting” below).

We recommend downloading `pong`’s visualization as a PNG or editable SVG. Users can change the image size of downloaded PNG in `pong` via a slider; depending on how the downloaded figure is displayed, a larger size PNG may appear to have higher-resolution.

## Troubleshooting:

- **PDF:** Users may need to disable pop-up blockers from `localhost` in order to have the print dialogs `pong` initiates for printing/downloading barplots work properly. Depending on the number of individuals in each  $Q$  matrix, the length of the population labels, and the user’s settings for printing from a given browser, the user may find that printed barplots are cut off in downloaded PDFs. There are two mechanisms by which this issue can be alleviated: (i) setting the margins to none in the print dialog; and/or (ii) resizing the browser window and then refreshing so that `pong` renders the visualization at a smaller dimension. If population labels are very large, then the labels may need to be shortened to avoid cropping in the print dialog.
- **PNG/SVG:** When downloading all barplots in `pong`’s main visualization and/or in a dialog box, if there are many values of  $K$  for which `pong` generated visualizations and/or many individuals, Google Chrome and Safari browsers may fail to fully download all barplots in the visualization. There are two workarounds: first, the Firefox web browser does not have this problem; second, Safari users can still download individual barplots.

## 9 Examples

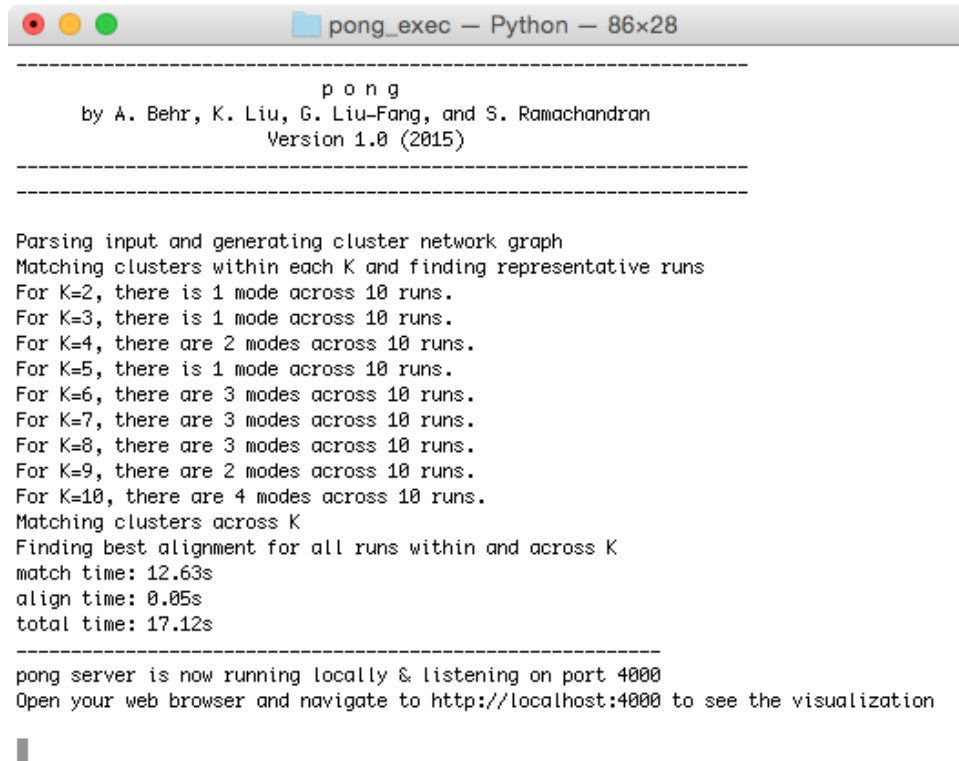
This section details some examples of running `pong`, and its resulting visualization.

### 9.1 1000 Genomes phase 3 dataset, main visualization

As mentioned in Section 4, we provide an example dataset of ADMIXTURE runs based on genotypes from the 1000 Genomes, phase 3 [[Consortium, 2015](#)], for `pong` users at [the Ramachandran Lab website](#).

To see `pong`’s visualization, run the following command in the example dataset directory `pong-example-data_1kG-p3`:

```
pong -m pong_filemap -i ind2pop.txt -n pop_order_expandednames.txt
```



```
pong_exec - Python - 86x28
-----
              p o n g
    by A. Behr, K. Liu, G. Liu-Fang, and S. Ramachandran
    Version 1.0 (2015)
-----

Parsing input and generating cluster network graph
Matching clusters within each K and finding representative runs
For K=2, there is 1 mode across 10 runs.
For K=3, there is 1 mode across 10 runs.
For K=4, there are 2 modes across 10 runs.
For K=5, there is 1 mode across 10 runs.
For K=6, there are 3 modes across 10 runs.
For K=7, there are 3 modes across 10 runs.
For K=8, there are 3 modes across 10 runs.
For K=9, there are 2 modes across 10 runs.
For K=10, there are 4 modes across 10 runs.
Matching clusters across K
Finding best alignment for all runs within and across K
match time: 12.63s
align time: 0.05s
total time: 17.12s
-----

pong server is now running locally & listening on port 4000
Open your web browser and navigate to http://localhost:4000 to see the visualization
```

Figure 1: Output from pong’s back end is displayed to the terminal as its algorithm runs. These messages detail information regarding modes identified for each value of  $K$ , as well as the time pong’s back end algorithms take for characterization of modes and alignment of  $Q$  matrices. Once the back end has completed processing the input  $Q$  matrices, pong prompts the user to open a browser window and connect to a specified host to initiate pong’s front end and display its visualization.

Figure 1 shows messages displayed to the user’s terminal window when the above command line is executed. All the specified files used as parameters in the above command line can be found in the example dataset directory, `pong-example-data_1kG-p3`. The required *filemap* (`pong_filemap`) is specified as input to `pong`. The particular command-line options used above also specify the population label for each individual in the  $Q$ -matrices (`ind2pop.txt`), and the population order with expanded names (`pop_order_expandednames.txt`).

Once the user is prompted to open a web-browser and navigate to `localhost:4000` (or the user-specified port; see Section 6.4.2), the main visualization is rendered in the browser window (Figure 2).

## 9.2 Chicken dataset from Rosenberg et al. [2001], visualizing modes

We also ran `pong` on 100  $Q$  matrices produced from running ADMIXTURE [Alexander et al., 2009] on 20 breeds of chickens at a single  $K$  value,  $K=19$ . Figure 3 shows pong’s dialog box

displaying the major mode, and some minor modes, for this dataset. `pong` identifies 28 modes in 15.91 minutes.

## References

- David H. Alexander, John Novembre, and Kenneth Lange. Fast model-based estimation of ancestry in unrelated individuals. Genome Research, 19(9):1655–1664, 2009.
- David M Blei, Andrew Y Ng, and Michael I Jordan. Latent Dirichlet Allocation. Journal of Machine Learning Research, 3(4-5):993–1022, 2003.
- The 1000 Genomes Project Consortium. A global reference for human genetic variation. Nature, 526(7571):68–74, 2015.
- Daniel Falush, Matthew Stephens, and Jonathan K Pritchard. Inference of population structure using multilocus genotype data: linked loci and correlated allele frequencies. Genetics, 164(4):1567–87, 2003.
- Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In Proceedings of the 7th Python in Science Conference (SciPy2008), pages 11–15, Pasadena, CA USA, 2008.
- Melissa J. Hubisz, Daniel Falush, Matthew Stephens, and Jonathan K. Pritchard. Inferring weak population structure with the assistance of sample group information. Molecular Ecology Resources, 9(5):1322–1332, 2009. doi: 10.1111/j.1755-0998.2009.02591.x.
- Mattias Jakobsson and Noah a Rosenberg. CLUMPP: a cluster matching and permutation program for dealing with label switching and multimodality in analysis of population structure. Bioinformatics (Oxford, England), 23(14):1801–6, July 2007.
- A Jasra, CC Holmes, and DA Stephens. Markov Chain Monte Carlo methods and the label switching problem in Bayesian Mixture Modeling. Statistical Science, 20(1):50–67, 2005.
- Naama M Kopelman, Jonathan Mayzel, Mattias Jakobsson, Noah A Rosenberg, and Itay Mayrose. C LUMPAK : a program for identifying clustering modes and packaging population structure inferences across K. Molecular Ecology Resources, pages doi: 10.1111/1755-0998.12387, 2015.
- Harold W Kuhn. The Hungarian Method for the assignment problem. Naval Research Logistics Quarterly, 2:83–97, 1955.
- Harold W Kuhn. Variants of the Hungarian method for assignment problems. Naval Research Logistics Quarterly, 3:253–258, 1956.
- Udi Manber. Introduction to Algorithms: A Creative Approach. Addison-Wesley, 1989. ISBN 0-201-12037-2.

- James Munkres. Algorithms for the Assignment and Transportation Problems. Journal of the Society of Industrial and Applied Mathematics, 5(1):32–38, 1957.
- J K Pritchard, M Stephens, and P Donnelly. Inference of population structure using multilocus genotype data. Genetics, 155(2):945–959, 2000.
- Anil Raj, Matthew Stephens, and Jonathan K Pritchard. fastSTRUCTURE: Variational Inference of Population Structure in Large SNP Data Sets. Genetics, 197(2):573–589, 2014.
- Noah a. Rosenberg. DISTRUCT: A program for the graphical display of population structure. Molecular Ecology Notes, 4(1):137–138, 2004.
- Noah A Rosenberg, Terry Burke, Kari Elo, Marcus W Feldman, Paul J Freidlin, Martien A M Groenen, Jossi Hillel, Asko Mäki-Tanila, Michèle Tixier-Boichard, Alain Vignal, Klaus Wimmers, and Steffen Weigend. Empirical evaluation of genetic clustering methods using multilocus genotypes from 20 chicken breeds. Genetics, 159(2):699–713, 2001.
- M Stephens. Dealing with label switching in mixture models. J. R. Statist. Soc. Series B, 62(4):795–809, 2000.

Figure 2: Screenshot of pong’s main visualization of population structure in the 1000 Genomes, based on our application of ADMIXTURE [Alexander et al., 2009] to these data; in order to fit this image on one page, we cropped the top of the browser window ( $K = 2, \dots, 5$ ). Each barplot depicts the representative run of the major at each value of  $K$ . Avg. similarity reports the average similarity between the representative run of a given major mode and all other runs in the major mode at that value of  $K$ . The number of runs in the major mode is displayed as a fraction to the left of each barplot. Clicking the printer icon to the left of a barplot initiates a print dialog for the chosen individual barplot in a separate web page. This allows the user to print (or save as PDF) any of the barplots in the visualization. For all  $K \geq 6$  (and  $K = 4$ , not shown here), a blue button appears to the right of the barplot, displaying the the number of additional modes pong detected at that value of  $K$ . Clicking on this will prompt a dialog box to open displaying all the minor modes at the given value of  $K$ . Population labels appear below the last barplot; the button labeled “Print all barplots” will print (or save to PDF) the full main visualization.

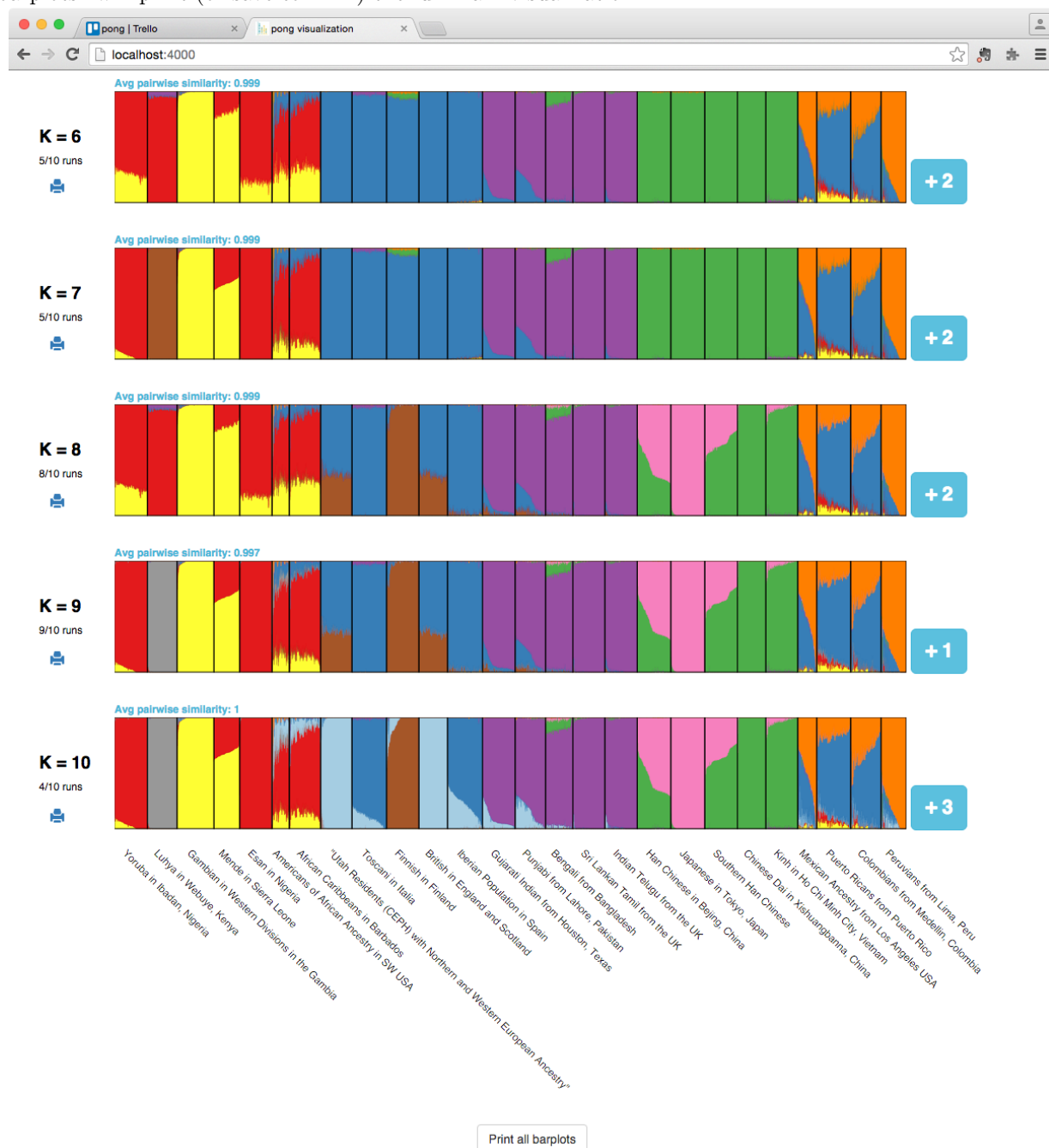




Figure 3: pong’s dialog box displaying major and 6 of 27 minor modes at  $K = 19$ , based on clustering inference output using data from chickens (analyzed initially by Rosenberg et al. [2001]; see also Figure 1 of Jakobsson and Rosenberg [2007]). pong will display all modes for a value of  $K$  if its back end algorithm identifies more than one mode at that value of  $K$ ; here our screenshot only shows 6 minor modes due to size constraints. We have checked the **Check to highlight multimodality** box; thus, clusters in each minor mode that are similar to the corresponding cluster in major mode are depicted in white, while maintaining full color for the dissimilar clusters when comparing each minor mode to the major mode. Printer icons also appear for each barplot in the dialog box; if barplots are printed, they will be printed as seen (in this case, with multimodality highlighted).

