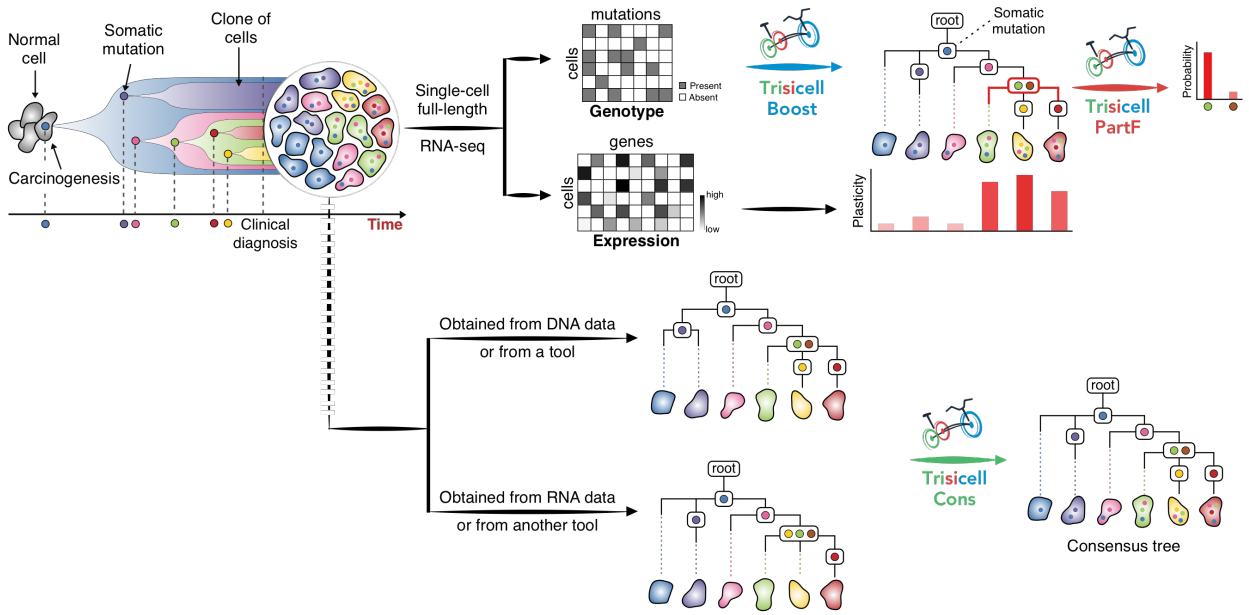
Trisicell documentation

Release master

National Cancer Institute

Dec 23, 2023

GENERAL



Trisicell (**Tri**ple-toolkit for **si**ngle-**cell** intratumor heterogeneity inference), pronounced as “tricycle”, is a new computational toolkit for scalable intratumor heterogeneity inference and evaluation from single-cell RNA, as well as single-cell genome or exome, sequencing data. Trisicell utilizes expressed SNVs and Indels to infer evolutionary relationships between genomic alterations and the cells that harbor them.

CHAPTER

ONE

SUPPORT

Feel free to submit an issue. Your help to improve Trisicell is highly appreciated.

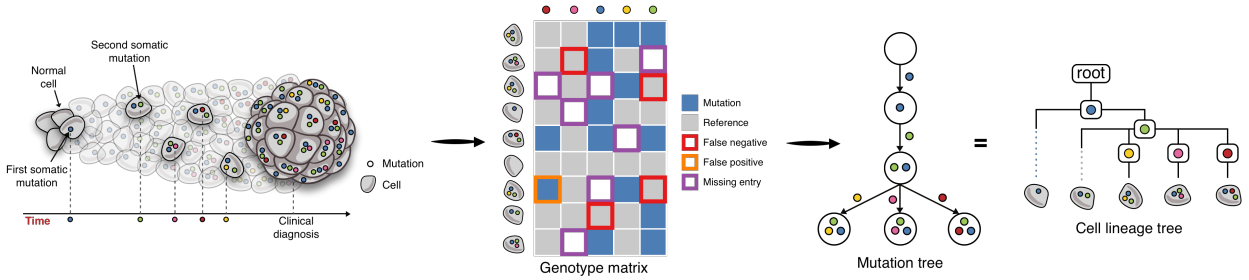
Trisicell was developed in collaboration between the Cancer Data Science Laboratory (CDSL) and the Laboratory of Cancer Biology and Genetics (LCBG) at the National Cancer Institute (NCI).

1.1 About Trisicell

Trisicell is a computational toolkit for scalable intratumor heterogeneity inference and validation from full-length transcriptome profiling of single-cell data (e.g. Smart-seq2) as well as single cell genome or exome sequencing data. Trisicell allows identifying and validating robust portions of a tumor progression tree, offering the ability to focus on the most important (sub)clones and the genomic alterations that seed the associated clonal expansion.

1.1.1 Tumor Progression Tree Reconstruction Models

Cancer is an evolutionary process. Looking back in time there were somatic mutational events that resulted in an emergence of the first cancerous cell. By the means of cell division, the number of cancerous cells grew, and newly born cells acquired additional mutations over time, which results in distinct populations of cells, each with its own complement of somatic mutations. Consequently, tumors are typically heterogeneous and are made of these different populations of cells. The heterogeneity provides the fuel for resistance either to the therapy or to the immune system. Single-cell sequencing (SCS) provides high resolution data that enables studying tumor progression tree and heterogeneity at unprecedented detail.



After mutation calling in every cell obtained by SCS, we have a matrix where rows represent individual cells and columns represent somatic mutations. The entries of the matrix show the presence/absence of mutations in cells. This matrix is better known as the genotype matrix. There are several types of noise present in the observed single-cell genotype matrix and they include:

- **False-negative errors** where a mutation is actually present in the cell but due to allele dropout (either biological or technical) or variance in sequencing coverage, in the genotype matrix it is reported as absent in the cell
- **False-positive errors** where a mutation is absent in the cell but in the genotype matrix it is reported to be present

- **Missing entries** where some entries contain no information about mutation presence or absence. Their presence in the matrix is usually due to insufficient coverage or lack of expression at the mutated loci in the cell

Due to the presence of false positive and false negative mutation calls, in most cases, the observed matrix contains a triplet of cells and a pair of columns such that in one of the three cells both mutations are reported to be present, in one of them it is reported that only the first mutation is present and in the remaining cell it is reported that only the second mutation is present. We say that such a triplet of cells and a pair of mutations form a *conflict*. These conflicts prevents us from reconstructing the tree of tumor progression directly from the observed matrix and requires the development of automated computational methods for finding the trees that best match the observed data.

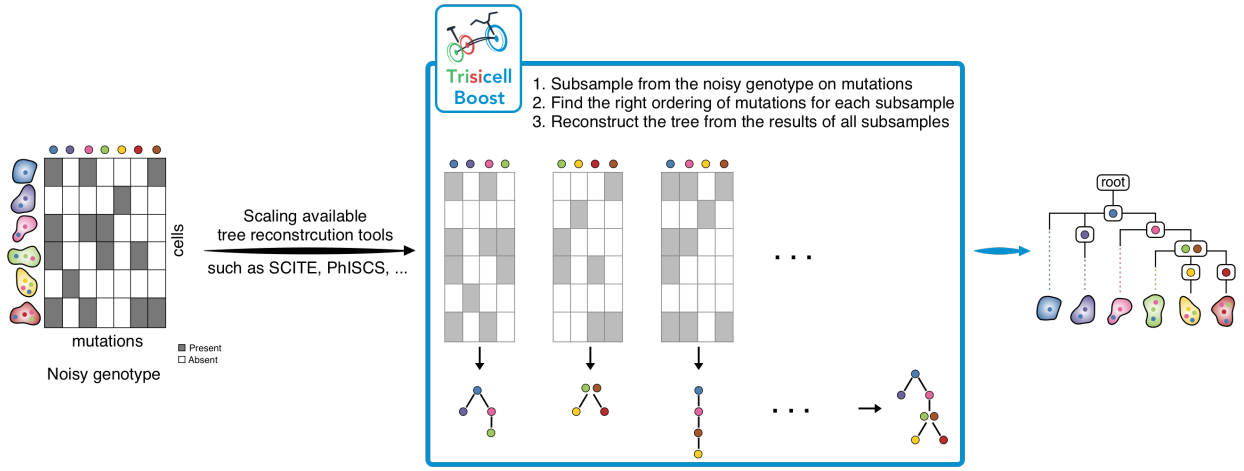
There are several techniques and methods to remove the noise/conflicts from the input genotype matrix. They are mostly based on Integer Linear Programming (ILP), Constraint Satisfaction Programming (CSP), Markov chain Monte Carlo (MCMC) sampling and Neighbor Joining (NJ). For more details, we highly recommend to read our Trisicell and review papers about building tumor progression tree by exploring the space of binary matrices.

1.1.2 Trisicell Components

Trisicell is comprised of three computational methods of independent but complementary aims and applications:

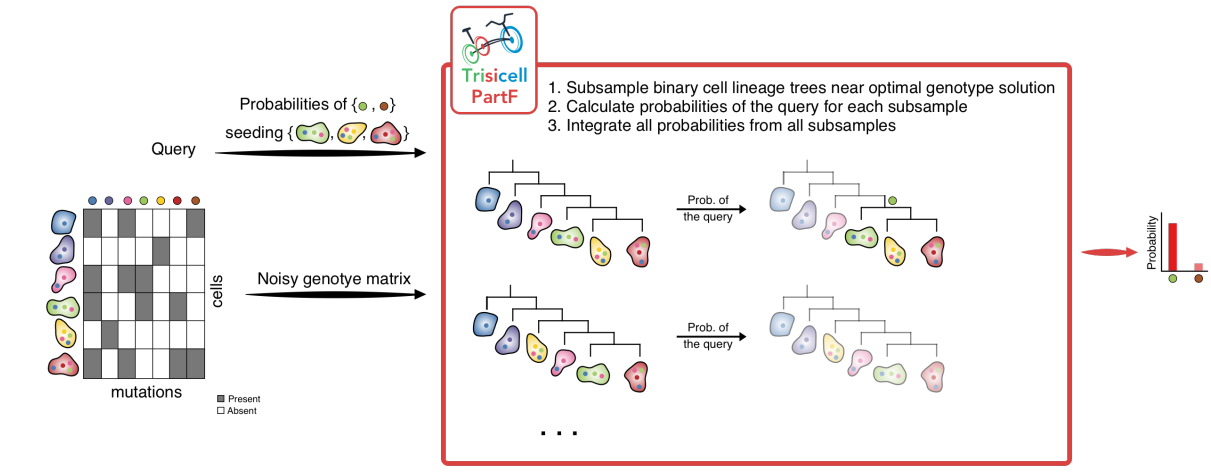
1) Trisicell-Boost:

Trisicell-Boost is a booster for phylogeny inference methods allowing them to scale up and handle large and noisy scRNAseq datasets.



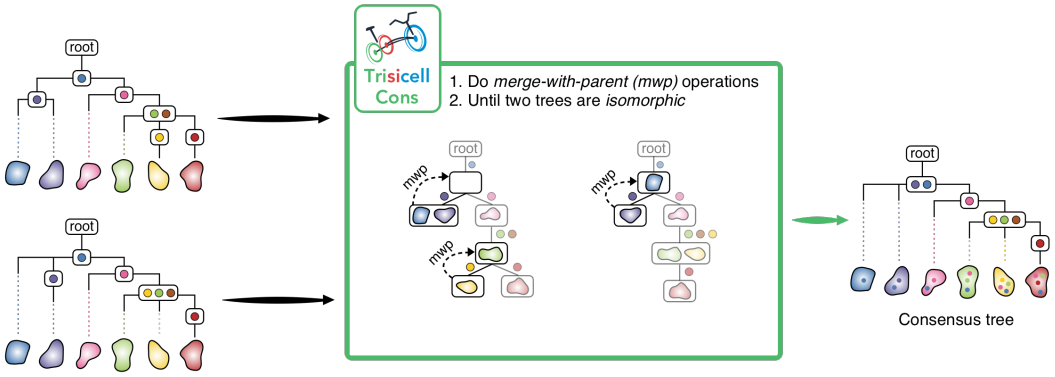
2) Trisicell-PartF:

Trisicell-PartF employs a localized sampling strategy to compute the probability of any user-specified set of cells forming a subclone seeded by one or more (again user- specified) mutations.



3) Trisicell-Cons:

Trisicell-Cons is devised to compare two or more phylogenies (more specifically cell-lineage trees) derived from the same single-cell data, and build their consensus tree by "collapsing" the minimum number of their edges so that the resulting trees are isomorphic



1.2 Installation

Trisicell requires Python 3.6 or later.

1.2.1 Using PyPI

Install trisicell from PyPI using:

```
pip install -U trisicell
```

`-U` is short for `--upgrade`. If you get a `Permission denied` error, use `pip install -U trisicell --user` instead.

1.2.2 Development Version

To work with the latest development version, install from GitHub using:

```
git clone https://github.com/faridrashidi/trisicell
cd trisicell
git fetch
git checkout develop
pip install -e '.[dev]'
```

`-e` stands for `--editable` and makes sure that your environment is updated when you pull new changes from GitHub. The `'[dev]'` options installs requirements needed for development, because Trisicell is bundled with an additional library.

Your contributions to improve Trisicell is highly appreciated! Please check out our contributing guide.

If you run into issues, do not hesitate to approach us or raise a GitHub issue.

1.3 Release Notes

1.3.1 Version 0.2.1 February 15, 2023

This version includes:

- Add treated mice data.
- Fix linting bug.

1.3.2 Version 0.2.0 January 27, 2023

This version includes:

- Add bWES data.
- Add bWTS data.
- Add scRNA data.
- Fix the bug of CI.

1.3.3 Version 0.1.1 December 28, 2022

This version includes:

- Fix the bug of CI.

1.3.4 Version 0.0.21 April 22, 2022

This version includes:

- Fix some bugs.
- Improve the documentations.

1.3.5 Version 0.0.20 November 22, 2021

This version includes:

- Add multi-threaded ScisTree.
- Update the documentations.

1.3.6 Version 0.0.19 October 18, 2021

This version includes:

- Add GRMT and SPhyR as new solvers.
- Update the consensus to new algorithm.
- Fix some bugs.

1.3.7 Version 0.0.18 October 13, 2021

This version includes:

- Add Robinson-Foulds metric into the score module.
- Fix the installation bug.

1.3.8 Version 0.0.17 September 29, 2021

This version includes:

- Fix the installation bug.

1.3.9 Version 0.0.16 September 27, 2021

This version includes:

- Add CASet and DISC for scoring two trees.
- Add mp3 for scoring two trees.
- Update dataset.
- Add more tests.
- Fix some bugs and typos.

1.3.10 Version 0.0.15 September 24, 2021

This version includes:

- Add consensus Day algorithm.
- Some bug fixes and typos.
- Add more tests.

1.3.11 Version 0.0.14 September 5, 2021

This version includes:

- Add more tests.

1.3.12 Version 0.0.13 September 2, 2021

This version includes:

- Lots of bug fixes and typos.
- Add more tests.
- Add more datasets.

1.3.13 Version 0.0.12 July 7, 2021

This version includes:

- Fix some bugs and typos.
- Add partition function command to the CLI.

1.3.14 Version 0.0.11 July 4, 2021

This version includes:

- Fix some typos.
- Fix some bugs.
- Fix SCITE cythonizing issue.

1.3.15 Version 0.0.10 July 3, 2021

This version includes:

- Fix some typos.
- Fix some bugs.

1.3.16 Version 0.0.9 June 17, 2021

This version includes:

- Fix some typos.
- Add flake8 support.

1.3.17 Version 0.0.8 June 12, 2021

This version includes:

- Trisicell-Boost function.
- A few more examples in the documentations.

1.3.18 Version 0.0.7 May 29, 2021

This version includes:

- PhISCS with readcount model.
- Cythonizing all the cpp files including SCITE, ScisTree and MLTD.
- Fix some bugs and typos.
- New datasets:
  - Leung et al., 2017 (colorectal cancer patient 1)

1.3.19 Version 0.0.6 May 25, 2021

This version includes:

- Add Stochastic Block Models (SBM) for sparse matrices.
- New datasets:
  - Hou et al., 2012 (myeloproliferative neoplasm).
  - Xu et al., 2012 (renal cell carcinoma).
  - Li et al., 2012 (muscle invasive bladder).
  - Wang et al., 2014 (oestrogen-receptor-positive breast cancer).
  - Wang et al., 2014 (triple-negative breast cancer).
  - Gawad et al., 2014 (acute lymphocytic leukemia patient 2).

1.3.20 Version 0.0.5 May 4, 2021

This version includes:

- Writing intermediate file in /tmp directory.
- Fix some bugs.

1.3.21 Version 0.0.4 April 17, 2021

This version includes:

- Add copy number tool.
- Fix some bugs.

1.3.22 Version 0.0.3 April 8, 2021

This version includes:

- Consensus tree builder with CLI command.
- Some new utility functions such as converting a tree fo conflict-free matrix.
- Bifiltering ILP code for selecting the maximal informed submatrix.

1.3.23 Version 0.0.2 March 29, 2021

Second beta release of Trisicell. This version includes:

- Solvers including (SCITE, PhISCS and etc).
- Preprocessing of the readcount matrices.
- Partition function estimation.
- Mutation calling commands for genotyping single-cell RNA data.
- Set of genotype noisy/solution datasets.
- Functions for comparing two clonal trees.
- Functions for plotting clonal/dendrogram trees.

1.3.24 Version 0.0.1 March 25, 2021

First beta release of Trisicell.

1.4 Citing

To cite Trisicell please use the following publication:

Profiles of expressed mutations in single cells reveal subclonal expansion patterns and therapeutic impact of intratumor heterogeneity Farid Rashidi Mehrabadi, Kerrie L. Marie, Eva Perez-Guijarro, Salem Malikic, Erfan Sadeqi Azer, Howard H. Yang, Can Kizilkale, Charli Gruen, Wells Robinson, Huaitian Liu, Michael C. Kelly, Christina Marcelus, Sandra Burkett, Aydin Buluc, Funda Ergun, Maxwell P. Lee, Glenn Merlino, Chi-Ping Day, S. Cenk Sahinalp bioRxiv 2021.03.26.437185; doi: https://doi.org/10.1101/2021.03.26.437185

1.4.1 BibTeX

```
@article{Trisicell,
  doi           = {10.1101/2021.03.26.437185},
  url           = {https://doi.org/10.1101/2021.03.26.437185},
  year          = 2021,
  month         = mar,
  publisher     = {Cold Spring Harbor Laboratory},
  author        = {Farid Rashidi Mehrabadi and Kerrie L. Marie and Eva Perez-Guijarro␣
↪and Salem Malikic and Erfan Sadeqi Azer and Howard H. Yang and Can Kizilkale and␣
↪Charli Gruen and Wells Robinson and Huaitian Liu and Michael C. Kelly and Christina␣
↪Marcelus and Sandra Burkett and Aydin Buluc and Funda Ergun and Maxwell P. Lee and␣
↪Glenn Merlino and Chi-Ping Day and S. Cenk Sahinalp},
  title         = {{Profiles of expressed mutations in single cells reveal subclonal␣
↪expansion patterns and therapeutic impact of intratumor heterogeneity}},
  journal       = {bioRxiv}
}
```

Trisicell Module.

1.5 API

Import Trisicell as:

```
import trisicell as tsc
```

After mutation calling and building the input data via our suggested *mutation calling pipeline*.

1.5.1 Read/Write (io)

This module offers a bunch of functions for reading and writing of the data.

io.read(filepath)	Read genotype matrix and read-count matrix.
io.write(obj, filepath)	Write genotype matrix or read-count matrix into a file.

trisicell.io.read

trisicell.io.**read**(*filepath*)

Read genotype matrix and read-count matrix.

The genotype matrix must be in the in format of pandas.DataFrame The read-count matrix must be in the format of anndata.AnnData.

Parameters **filepath** (str) – The path to the file. The extension must be one of [.tsv, .SC, .CFMatrix, .h5ad, .h5ad.gz, .nwk]

Returns Depends on the format of the input file the output type is different.

Return type pandas.DataFrame or anndata.AnnData

trisicell.io.write

trisicell.io.**write**(*obj*, *filepath*)

Write genotype matrix or read-count matrix into a file.

Parameters

- **obj** (pandas.DataFrame or anndata.AnnData) – The input object which is going to be written in a file.
- **filepath** (str) – The file path where the obj must be written in.

1.5.2 Preprocessing (pp)

This module offers a bunch of functions for filtering and preprocessing of the data.

pp.remove_mut_by_list(adata, alist)	Remove a list of mutations from the data.
pp.remove_cell_by_list(adata, alist)	Remove a list of cells from the data.
pp.filter_mut_reference_must_present_in_at_le	Remove mutations based on the wild-type status.
pp.filter_mut_mutant_must_present_in_at_leas	Remove mutations based on the mutant status.
pp.consensus_combine(df)	Combine cells in genotype matrix.

trisicell.pp.remove_mut_by_list

trisicell.pp.**remove_mut_by_list**(*adata*, *alist*)

Remove a list of mutations from the data.

Parameters

- **adata** (anndata.AnnData) – The input readcount data.
- **alist** (list) – A list of mutations.

trisicell.pp.remove_cell_by_list

trisicell.pp.**remove_cell_by_list**(*adata*, *alist*)

Remove a list of cells from the data.

Parameters

- **adata** (anndata.AnnData) – The input readcount data.
- **alist** (list) – A list of cells.

trisicell.pp.filter_mut_reference_must_present_in_at_least

trisicell.pp.**filter_mut_reference_must_present_in_at_least**(*adata*, *min_cells=1*)

Remove mutations based on the wild-type status.

This function removes mutations that don't have the status of wild-type in at least `min_cells` cells. Note that mutations that are present in all cells will not be filtered out by this function.

Parameters

- **adata** (anndata.AnnData) – The input readcount data.
- **min_cells** (int, optional) – Minimum number of cells, by default 1

trisicell.pp.filter_mut_mutant_must_present_in_at_least

trisicell.pp.**filter_mut_mutant_must_present_in_at_least**(*adata*, *min_cells=2*)

Remove mutations based on the mutant status.

This function removes mutations that don't have the status of mutant in at least `min_cells` cells.

Parameters

- **adata** (anndata.AnnData) – The input readcount data.
- **min_cells** (int, optional) – Minimum number of cells, by default 1

trisicell.pp.consensus_combine

trisicell.pp.**consensus_combine**(*df*)

Combine cells in genotype matrix.

This function combines the replicates or cells that have the same origin prior to running Trisicell-Cons. The replicates or cells that are supposed to be merged must be designated with _. For instance:

input: {{Cell1}_{ID1}, {Cell1}_{ID2}, {Cell2}_{ID1}, {Cell2}_{ID2}}.

output: {{Cell1}, {Cell2}}.

Parameters **df** (pandas.DataFrame) – The input genotype matrix in conflict-free format.

Returns The combine genotype matrix.

Return type pandas.DataFrame

1.5.3 Tools (tl)

This module offers a high-level API to compute the conflict-free solution and calculating the probability of mutations seeding particular cells.

Solving the noisy input genotype matrix (Trisicell-Boost)

tl.booster(df_input, alpha, beta[, solver, ...])	Trisicell-Boost solver.

trisicell.tl.booster

trisicell.tl.**booster**(*df_input*, *alpha*, *beta*, *solver='SCITE'*, *sample_on='muts'*, *sample_size=10*, *n_samples=10*, *begin_index=0*, *n_jobs=10*, *dep_weight=50*, *time_limit=120*, *n_iterations=500000*, *subsample_dir=None*, *disable_tqdm=False*, *no_subsampling=False*, *no_dependencies=False*, *no_reconstruction=False*)

Trisicell-Boost solver.

For more details of available tools that work on binary matrices, read [ReviewBinary].

Parameters

- **df_input** (pandas.DataFrame) – input noisy dataframe
- **alpha** (float) – false positive rate
- **beta** (float) – false negative rate
- **solver** (str, optional) – which tool is boosted {"SCITE", "PhISCS"}, by default "SCITE"
- **sample_on** (str, optional) – on which dimension is subsampled {"muts", "cells"}, by default "muts"
- **sample_size** (int, optional) – number of subsampled mutations or cells depends on sample_on, by default 10
- **n_samples** (int, optional) – number of samples, by default 10
- **begin_index** (int, optional) – start index of intermediate file names, by default 0
- **n_jobs** (int, optional) – number of jobs, by default 10
- **dep_weight** (int, optional) – weight multiplier, by default 50
- **time_limit** (int, optional) – time out needed for PhISCS running on each instance, by default 120
- **n_iterations** (int, optional) – number of iterations needed for SCITE running, by default 500000
- **subsample_dir** (str, optional) – for keeping the intermediate subsamples CFMatrices, by default None
- **disable_tqdm** (bool, optional) – disable progress bar, by default False
- **no_subsampling** (bool, optional) – subsampling (step 1/3) gets off, by default False
- **no_dependencies** (bool, optional) – dependencies calculation (step 2/3) gets off, by default False
- **no_reconstruction** (bool, optional) – reconstruction of big tree (step 3/3) gets off, by default False

Returns A conflict-free matrix in which rows are cells and columns are mutations. Values inside this matrix show the presence (1) and absence (0).

Return type pandas.DataFrame

See also:

trisicell.tl.scite(), trisicell.tl.phiscsb()

Examples

- *Construct lienage tree using Trisicell-Boost*

Partition function calculation (Trisicell-PartF)

tl.partition_function(df_input, alpha, beta, ...)	Calculate the probability of a mutation seeding particular cells.

trisicell.tl.partition_function

trisicell.tl.**partition_function**(*df_input*, *alpha*, *beta*, *n_samples*, *n_batches*, *muts*, *cells*)

Calculate the probability of a mutation seeding particular cells.

Parameters

- **df_input** (pandas.DataFrame) – Input genotype matrix.
- **alpha** (*float*) – False positive error rate.
- **beta** (*float*) – False negative error rate.
- **n_samples** (*int*) – Number of samples to get from the distribution (suggest: 1000)
- **n_batches** (*int*) – Number of batches to repeat the experiment (suggest: 100)
- **muts** (*list*) – The list of mutations
- **cells** (*list*) – The list of cells

Returns A table of probabilities for every mutation and every batch.

Return type pandas.DataFrame

Consensus tree building (Trisicell-Cons)

tl.consensus(sc1, sc2)	Build the consensus tree between two tumor progression trees.

trisicell.tl.consensus

trisicell.tl.**consensus**(*sc1*, *sc2*)

Build the consensus tree between two tumor progression trees.

Parameters

- **df1** (pandas.DataFrame) – First conflict-free matrix.
- **df2** (pandas.DataFrame) – Second conflict-free matrix.

Returns The consensus tree in which cells are in edges and mutations are in edges.

Return type networkx.DiGraph

1.5.4 Plotting (pl)

This module offers plotting the tree in clonal or dendrogram format.

pl.clonal_tree(tree[, muts_as_number, ...])	Draw the tree in clonal format.
pl.dendro_tree(tree[, width, height, dpi, ...])	Draw the tree in dendro fromat.

trisicell.pl.clonal_tree

trisicell.pl.**clonal_tree**(*tree*, *muts_as_number=False*, *cells_as_number=False*, *show_id=False*, *cell_info=None*, *output_file=None*, *color_attr=None*, *dpi=150*)

Draw the tree in clonal format.

This functions plots the tree in which edges are mutations and nodes are cells.

Parameters

- **tree** (networkx.DiGraph) – The input tree.
- **muts_as_number** (bool, optional) – Change the mutation list to a number at edges, by default False
- **cells_as_number** (bool, optional) – Change the cell list to a number at edges, by default False
- **show_id** (bool, optional) – Whether to show IDs of nodes and edges or not, by default True
- **cell_info** (pandas.DataFrame, optional) – Information of cells for coloring the nodes by a pie chart, by default None
- **output_file** (str, optional) – Path to a file for saving the tree in, by default None
- **color_attr** (str, optional) – Attributes in the cell_info dataframe for coloring the nodes, by default None
- **dpi** (int, optional) – Resolution of rendered figures – this influences the size of figures in notebooks, by default 150

Returns

Return type None

trisicell.pl.dendro_tree

trisicell.pl.**dendro_tree**(*tree*, *width=1200*, *height=500*, *dpi=300*, *cell_info=None*, *label_color='group_color'*, *line_size=0.4*, *tiplab_size=2.5*, *inner_node_type='nmuts'*, *inner_node_size=2*, *distance_labels_to_bottom=4*, *annotation=None*, *output_file=None*)

Draw the tree in dendro fromat.

Parameters

- **tree** (networkx.DiGraph) – The input tree.
- **width** (int, optional) – Width of the figure, by default 8
- **height** (int, optional) – Height of the figure, by default 3

- **dpi** (int, optional) – The resolution, by default 300
- **cell_info** (pandas.DataFrame, optional) – Information about cells such as color, expression values of genes and etc, by default None
- **label_color** (str, optional) – The column name in which colors of cells are stored in the dataframe provided as `cell_info`, by default "group_color"
- **line_size** (float, optional) – Line size of the tree, by default 0.4
- **tiplab_size** (float, optional) – Cell name size in the tree, by default 2.5
- **inner_node_type** (str, optional) – The format of the inner nodes (i.e. mutations), by default "nmuts" Values are:
  - `nmuts`: only number of mutations
  - `nodeid`: only node id
  - `both`: both number of mutations and node id
- **inner_node_size** (int, optional) – Size of the inner nodes (i.e. mutations), by default 2
- **distance_labels_to_bottom** (int, optional) – Distance cell names to the bottom of the figure, by default 4
- **annotation** (list, optional) – List of gene names provided in the column dataframe of `cell_info` in to be annotated in the bottom of the tree, by default []
- **output_file** (str, optional) – Path to a file for saving the tree in, by default None

Returns

Return type None

Note: The cell names in the tree must be identical to the index of `cell_info` dataframe if it was provided.

1.5.5 Utils (ul)

This module offers a bunch of utility functions.

ul.to_tree(df)	Convert a conflict-free matrix to a tree object.
ul.to_cfmatrix(tree)	Convert phylogenetic tree to conflict-free matrix.
ul.to_mtree(tree)	Convert the phylogenetic tree to mutation tree.

trisicell.ul.to_tree

trisicell.ul.**to_tree**(*df*)

Convert a conflict-free matrix to a tree object.

This function converts a conflict-free matrix to a tree object in which nodes are labled with cells and edges are lables with mutations. The root is labled by 'root'. Mutations are seperated by `.graph['splitter_mut']` and cells are seperated by `.graph['splitter_cell']`. Those mutations that are not present in any cell are stored in `.graph['become_germline']`. Mutations happed once during the evolution so there is no repetitive mutation.

Parameters **df** (pandas.DataFrame) – A genotype dataframe in which rows are cells and columns are mutations. Note that this dataframe must be conflict-free.

Returns A perfect phylogenetic tree.

Return type networkx.DiGraph

trisicell.ul.to_cfmatrix

trisicell.ul.**to_cfmatrix**(*tree*)

Convert phylogenetic tree to conflict-free matrix.

This function converts a phylogenetic tree in which mutations are at edges and cells are at nodes to a conflict-free matrix where rows are cells, columns are mutations and each entry is either zero or one representing the absence or presence of the mutation in the cell.

Parameters **tree** (networkx.DiGraph) – The phylogenetic tree.

Returns The conflict-free matrix.

Return type pandas.DataFrame

trisicell.ul.to_mtree

trisicell.ul.**to_mtree**(*tree*)

Convert the phylogenetic tree to mutation tree.

Parameters **tree** (networkx.DiGraph) – The phylogenetic tree in which cells are in nodes and mutations are at edges.

Returns The mutation tree in which mutations are in nodes.

Return type networkx.DiGraph

1.5.6 Datasets (datasets)

This module offers a bunch of functions for simulating data.

datasets.example([is_expression])	Return an example for sanity checking and playing with Trisicell.
datasets.sublines_bwes()	Trisicell sublines bWES data.
datasets.sublines_bwts()	Trisicell sublines bWTS data.
datasets.sublines_scrnaseq()	Trisicell sublines scRNAseq data.
datasets.treated_actla4()	Trisicell treated mice (anti-ctla-4) scRNAseq data.
datasets.treated_igg_ss2()	Trisicell treated mice (igg, smart-seq2) scRNAseq data.
datasets.treated_igg_sw()	Trisicell treated mice (igg, seq-well) scRNAseq data.

trisicell.datasets.example

trisicell.datasets.**example**(*is_expression=False*)

Return an example for sanity checking and playing with Trisicell.

is_expression [bool, optional] Returns the expression dataset instead of the genotype one, by default False

Returns An object that cells are in `.obs` and mutations are in `.var`.

Return type anndata.AnnData

trisicell.datasets.sublines_bwes

trisicell.datasets.**sublines_bwes**()
Trisicell sublines bWES data.

The size is n_sublines × n_muts = 24 × 6653

Returns

An anndata in which `.var` contains information about the mutations.

- **`.layers['trisicell_input']` the binary input genotype matrix used as** input to the Trisicell.
- **`.layers['trisicell_output']` the binary input genotype matrix inferred by** Trisicell-boost(SCITE).
- **`.layers['genotype']` noisy genotype matrix, 0: reference, 1: heterozygous** 2: unknown and 3: homozygous_alt.
- `.layers['mutant']` number of mutant reads.
- `.layers['total']` number of total reads.

Return type anndata.AnnData

Examples

```
>>> adata = tsc.datasets.sublines_bwes()
>>> adata
AnnData object with n_obs × n_vars = 24 × 6653
    var: 'kind', 'amino_acid_change', 'ensemble', 'gene', 'chrom', 'position', ...
    layers: 'genotype', 'mutant', 'total', 'trisicell_input', 'trisicell_output'
```

See also:

trisicell.datasets.sublines_scrnaseq(), *trisicell.datasets.sublines_bwts()*

trisicell.datasets.sublines_bwts

trisicell.datasets.**sublines_bwts**()
Trisicell sublines bWTS data.

The size is n_cells × n_muts = 33 × 536

Returns A mudata with two modalities (`.mod`)

Return type mudata.MuData

Examples

```
>>> mdata = tsc.datasets.sublines_bwts()
>>> mdata
MuData object with n_obs × n_vars = 33 × 55851
2 modalities
  expression: 175 x 55401
    obs:    'cells', 'uniquely_mapped_percent', 'num_splices', ...
    layers: 'fpkm', 'tpm'
  mutation: 33 x 536
    obs:    'cells', 'clone', 'mps', 'zscore', 'group', 'Axl', 'Mitf', 'day', ...
    var:    'kind', 'amino_acid_change', 'ensemble', 'gene', 'chrom', 'position',...
    layers: 'genotype', 'mutant', 'total', 'trisicell_input', 'trisicell_output'
```

See also:

trisicell.datasets.sublines_bwes(), *trisicell.datasets.sublines_scrnaseq()*

trisicell.datasets.sublines_scrnaseq

trisicell.datasets.**sublines_scrnaseq**()

Trisicell sublines scRNAseq data.

The size is n_cells × n_muts = 175 × 450

Returns A mudata with two modalities (`.mod`)

Return type mudata.MuData

Examples

```
>>> mdata = tsc.datasets.sublines_scrnaseq()
>>> mdata
MuData object with n_obs × n_vars = 175 × 55851
2 modalities
  expression: 175 x 55401
    obs:    'cells', 'uniquely_mapped_percent', 'num_splices', ...
    layers: 'fpkm', 'tpm'
  mutation: 175 x 450
    obs:    'cells', 'clone', 'group', 'group_color', 'is_red', 'is_sub', ...
    var:    'kind', 'amino_acid_change', 'ensemble', 'gene', 'chrom', ...
    layers: 'genotype', 'mutant', 'total', 'trisicell_input', 'trisicell_output'
```

See also:

trisicell.datasets.sublines_bwes(), *trisicell.datasets.sublines_bwts()*

trisicell.datasets.treated_actla4

trisicell.datasets.**treated_actla4**()
Trisicell treated mice (anti-ctla-4) scRNAseq data.

The size is n_cells × n_muts = 508 × 3309

Returns A mudata with two modalities (.mod)

Return type mudata.MuData

Examples

```
>>> mdata = tsc.datasets.treated_actla4()
>>> mdata
MuData object with n_obs × n_vars = 508 × 58710
2 modalities
  mutation: 508 x 3309
    obs:    'group', 'Mouse_ID', 'Batch', 'Axl', 'Erbb3', 'Mitf', 'H2-K1', ...
    var:    'kind', 'amino_acid_change', 'ensemble', 'gene', 'chrom', ...
    uns:    'trisicell_input', 'trisicell_output'
    layers: 'genotype', 'mutant', 'total'
  expression:       508 x 55401
    obs:    'uniquely_mapped_percent', 'num_splices', 'num_GCAG_splices', ...
    layers: 'fpkm', 'tpm'
```

See also:

trisicell.datasets.treated_igg_ss2(), *trisicell.datasets.treated_igg_sw()*

trisicell.datasets.treated_igg_ss2

trisicell.datasets.**treated_igg_ss2**()
Trisicell treated mice (igg, smart-seq2) scRNAseq data.

The size is n_cells × n_muts = 163 × 1453

Returns A mudata with two modalities (.mod)

Return type mudata.MuData

Examples

```
>>> mdata = tsc.datasets.treated_igg_ss2()
>>> mdata
MuData object with n_obs × n_vars = 163 × 56854
2 modalities
  mutation: 163 x 1453
    obs:    'group', 'Mouse_ID', 'Batch', 'Axl', 'Erbb3', 'Mitf', 'H2_K1', ...
    var:    'kind', 'amino_acid_change', 'ensemble', 'gene', 'chrom', ...
    uns:    'trisicell_input', 'trisicell_output'
    layers: 'genotype', 'mutant', 'total'
  expression:       163 x 55401
```

(continues on next page)

(continued from previous page)

```
    obs:    'uniquely_mapped_percent', 'num_splices', 'num_GCAG_splices', ...
    layers: 'fpkm', 'tpm'
```

See also:

trisicell.datasets.treated_actla4(), *trisicell.datasets.treated_igg_sw()*

trisicell.datasets.treated_igg_sw

trisicell.datasets.**treated_igg_sw**()

Trisicell treated mice (igg, seq-well) scRNAseq data.

The size is n_cells × n_muts = 163 × 1453

Returns A mudata with two modalities (`.mod`)

Return type mudata.MuData

Examples

```
>>> mdata = tsc.datasets.treated_igg_sw()
>>> mdata
MuData object with n_obs × n_vars = 163 × 56854
2 modalities
  mutation: 163 x 1453
    obs:    'group', 'Mouse_ID', 'Batch', 'Axl', 'Erbb3', 'Mitf', 'H2_K1', ...
    var:    'kind', 'amino_acid_change', 'ensemble', 'gene', 'chrom', ...
    uns:    'trisicell_input', 'trisicell_output'
    layers: 'genotype', 'mutant', 'total'
  expression:       163 x 55401
    obs:    'uniquely_mapped_percent', 'num_splices', 'num_GCAG_splices', ...
    layers: 'fpkm', 'tpm'
```

See also:

trisicell.datasets.treated_actla4(), *trisicell.datasets.treated_igg_ss2()*

1.6 CLI

A command line interface (CLI) is available in Trisicell package. After you have trisicell correctly installed on your machine (see *installation tutorial*), the `trisicell` command will become available in the terminal. `trisicell` is a command line tool with subcomands. You can get quick info on all the available commands typing `trisicell --help`. You will get the following output:

```
Usage: trisicell [OPTIONS] COMMAND [ARGS]...

  Trisicell.

  Scalable intratumor heterogeneity inference and validation from single-cell data.

Options:
```

(continues on next page)

(continued from previous page)

```
  --version  Show the version and exit.
  --help     Show this message and exit.

Commands:
  mcalling   Mutation calling.
  booster    Boost available tree reconstruction tool (Trisicell-Boost).
  partf      Get samples or calculate for PartF.
  consensus  Build consensus tree between two phylogenetic trees (Trisicell-Cons).
```

1.6.1 `mcalling` - Run Mutation Calling

trisicell

Trisicell.

Scalable intratumor heterogeneity inference and validation from single-cell data.

```
trisicell [OPTIONS] COMMAND [ARGS]...
```

Options

--version
Show the version and exit.

mcalling

Mutation calling pipeline.

trisicell mcalling config.yml

A template of the `config file` can be found here.

```
trisicell mcalling [OPTIONS] CONFIG_FILE
```

Options

-t, --test
Is in test mode.

Default `False`

-s, --status
Check the status of runs.

Default `False`

-b, --build
Build the final matrices of genotype and expression.

Default `False`

-c, --clean
Cleaning the directory including removing intermediate files.

Default `False`

Arguments

CONFIG_FILE
Required argument

1.6.2 `booster` - Run Booster

trisicell

Trisicell.

Scalable intratumor heterogeneity inference and validation from single-cell data.

```
trisicell [OPTIONS] COMMAND [ARGS]...
```

Options

--version
Show the version and exit.

booster

Boost available tree reconstruction tool.

For doing all 3 steps:

trisicell booster input.SC 0.001 0.1 –solver scite –n_samples 200 –sample_size 15 –n_jobs 4 –n_iterations 10000 –dep_weight 50 –subsample_dir . –begin_index 0

For doing only the last step:

trisicell booster input.SC 0.001 0.1 –dep_weight 50 –subsample_dir PATH_TO_SUBSAMPLES_FOLDER –no_subsampling –no_dependencies

```
trisicell booster [OPTIONS] GENOTYPE_FILE ALPHA BETA
```

Options

--solver <solver>
Solver of the booster.

Default `scite`

Options scite | phiscs | scistree

--sample_on <sample_on>
Sampling on `muts` or `cells`.

Default `muts`

Options muts | cells

--sample_size <sample_size>
The size of samples i.e. the number of muts or cells.

Default 10

--n_samples <n_samples>
The number of subsamples.

Default 10

--begin_index <begin_index>
ID of the start subsample name.

Default 0

--n_jobs <n_jobs>
Number of parallel jobs to do subsampleing.

Default 0

--dep_weight <dep_weight>
Weight for how many subsamples to be used in dependencies calculation.

Default 50

--time_limit <time_limit>
Timeout of solving allowance (in second).

Default 120

--n_iterations <n_iterations>
SCITE number of iterations.

Default 500000

--subsample_dir <subsample_dir>
A path to the subsamples directory.

--disable_tqdm
Disable showing the tqdm progress.

Default False

--no_subsampling
No running of subsampling (step 1/3).

Default False

--no_dependencies
No running of subsampling (step 2/3).

Default False

--no_reconstruction
No running of reconstruction (step 3/3).

Default False

Arguments

GENOTYPE_FILE
Required argument

ALPHA
Required argument

BETA
Required argument

1.6.3 consensus - Run Consensus

trisicell

Trisicell.

Scalable intratumor heterogeneity inference and validation from single-cell data.

```
trisicell [OPTIONS] COMMAND [ARGS]...
```

Options

--version
Show the version and exit.

consensus

Build consensus tree between two phylogenetic trees.

It writes the conflict-free matrix representing the consensus tree into the `consensus_path` filepath.

trisicell consensus first_tree.CFMatrix second_tree.CFMatrix

```
trisicell consensus [OPTIONS] FIRST_TREE SECOND_TREE CONSENSUS_PATH
```

Arguments

FIRST_TREE
Required argument

SECOND_TREE
Required argument

CONSENSUS_PATH
Required argument

1.7 Mutation Calling

The overview of the mutation calling pipeline that was used in Trisicell is provided below.

We provide a more detailed description of each of these steps.

1.7.1 Step 1:

```
STAR \
    --runMode genomeGenerate \
    --genomeDir {outdir}/{indexing_1} \
    --genomeFastaFiles {ref} \
    --sjdbGTFfile {gtf_file} \
    --sjdbOverhang {readlength} \
    --runThreadN {number_of_threads}
```

1.7.2 Step 2:

```
# for every `cell_id`
trim_galore \
    --gzip \
    --length 30 \
    --fastqc \
    --output_dir {outdir}/{cell_id} \
    --paired \
    {cell_id}/{fastq_file_1} {cell_id}/{fastq_file_2}

# for every `cell_id`
STAR \
    --runMode alignReads \
    --genomeDir {outdir}/{indexing1} \
    --sjdbGTFfile {gtf_file} \
    --outFilterMultimapNmax 1 \
    --outSAMunmapped None \
    --quantMode TranscriptomeSAM GeneCounts \
    --runThreadN 1 \
    --sjdbOverhang {read_length} \
    --readFilesCommand zcat \
    --outFileNamePrefix {outdir}/{cell_id}/ \
    --readFilesIn {cell_id}/{fastq_file_1_trim_galore} {cell_id}/{fastq_file_2_trim_
↪galore}
```

1.7.3 Step 3:

```
STAR \
    --runMode genomeGenerate \
    --genomeDir {outdir}/{indexing_2} \
    --genomeFastaFiles {ref} \
    --sjdbGTFfile {gtf_file} \
    --sjdbFileChrStartEnd {all_cell_files [*SJ.out.tab]} \
    --sjdbOverhang {read_length} \
    --runThreadN {number_of_threads}
```

1.7.4 Step 4:

```
# for every `cell_id`
STAR \
    --runMode alignReads \
    --genomeDir {outdir}/{indexing_2} \
    --readFilesCommand zcat \
    --readFilesIn {cell_id}/{fastq_file_1_trim_galore} {cell_id}/{fastq_file_2_trim_
↪galore} \
    --outFileNamePrefix {outdir}/{cell_id}/ \
    --limitBAMsortRAM 300000000000 \
    --outSAMtype BAM SortedByCoordinate \
    --sjdbGTFfile {gtf_file} \
    --outFilterMultimapNmax 1 \
    --outSAMunmapped None \
    --quantMode TranscriptomeSAM GeneCounts \
    --runThreadN 1 \
    --sjdbOverhang {read_length}

# for every `cell_id`
java -Xmx90g -jar PICARD.jar \
    AddOrReplaceReadGroups \
    INPUT={outdir}/{cell_id}/Aligned.sortedByCoord.out.bam \
    OUTPUT={outdir}/{cell_id}/dedupped.bam \
    SORT_ORDER=coordinate \
    RGID={cell_id} \
    RGLB=trancriptome \
    RGPL=ILLUMINA \
    RGPU=machine \
    RGSM={cell_id}

# for every `cell_id`
java -Xmx90g -jar PICARD.jar \
    MarkDuplicates \
    INPUT={outdir}/{cell_id}/dedupped.bam \
    OUTPUT={outdir}/{cell_id}/rg_added_sorted.bam \
    METRICS_FILE={outdir}/{cell_id}/output.metrics \
    VALIDATION_STRINGENCY=SILENT \
    CREATE_INDEX=true
```

(continues on next page)

(continued from previous page)

```
# for every `cell_id`
java -Xmx90g -jar GATK.jar \
    -T SplitNCigarReads \
    -R {ref} \
    -I {outdir}/{cell_id}/rg_added_sorted.bam \
    -o {outdir}/{cell_id}/split.bam \
    -rf ReassignOneMappingQuality \
    -RMQF 255 \
    -RMQT 60 \
    -U ALLOW_N_CIGAR_READS

# for every `cell_id`
java -Xmx90g -jar GATK.jar \
    -T RealignerTargetCreator \
    -R {ref} \
    -I {outdir}/{cell_id}/split.bam \
    -o {outdir}/{cell_id}/indel.intervals \
    -known {db_snps_indels} \
    -U ALLOW_SEQ_DICT_INCOMPATIBILITY

# for every `cell_id`
java -Xmx90g -jar GATK.jar \
    -T IndelRealigner \
    -R {ref} \
    -I {outdir}/{cell_id}/split.bam \
    -o {outdir}/{cell_id}/realign.bam \
    -targetIntervals {outdir}/{cell_id}/indel.intervals \
    -known {db_snps_indels}

# for every `cell_id`
java -Xmx90g -jar GATK.jar \
    -T BaseRecalibrator \
    -R {ref} \
    -I {outdir}/{cell_id}/realign.bam \
    -o {outdir}/{cell_id}/recal.table \
    -knownSites {db_snps_indels}

# for every `cell_id`
java -Xmx90g -jar GATK.jar \
    -T PrintReads \
    -R {ref} \
    -I {outdir}/{cell_id}/realign.bam \
    -o {outdir}/{cell_id}/output.bam \
    -BQSR {outdir}/{cell_id}/recal.table
```

1.7.5 Step 5:

```
# for every `cell_id`
java -Xmx90g -jar GATK.jar \
    -T HaplotypeCaller \
    -R {ref} \
    -I {outdir}/{cell_id}/output.bam \
    -o {outdir}/{cell_id}/HaplotypeCaller.g.vcf \
    -dontUseSoftClippedBases \
    -stand_call_conf 20 \
    --dbsnp {db_snps_indels} \
    -ERC GVCF
```

1.7.6 Step 6:

```
java -Xmx90g -jar GATK.jar \
    -T GenotypeGVCFs \
    -R {ref} \
    -V {all_cell_files [*HaplotypeCaller.g.vcf]} \
    -o {outdir}/jointcalls.g.vcf \
    -nt {number_of_threads} \
    --disable_auto_index_creation_and_locking_when_reading_rods
```

1.8 Building Phylogeny

First we import the Trisicell pakcage as:

[1]:
```
import trisicell as tsc

tsc.settings.verbosity = 3  # show errors(0), warnings(1), info(2), hints(3)
tsc.logg.print_version()
```

```
Running trisicell 0.0.13 (python 3.7.10) on 2021-09-03 10:26.
```

[2]:
```
adata = tsc.datasets.example()
```

[3]:
```
adata
```

[3]:
```
AnnData object with n_obs × n_vars = 83 × 452
    obs: 'group', 'subclone_color', 'Axl', 'Erbb3', 'Mitf', 'MPS'
    var: 'CHROM', 'POS', 'REF', 'ALT', 'START', 'END', 'Allele', 'Annotation', 'Gene_Name
↪', 'Transcript_BioType', 'HGVS.c', 'HGVS.p'
    layers: 'genotype', 'mutant', 'total'
```

Here is the information about the cells:

[4]:
```
adata.obs
```

[4]:
```
      group subclone_color       Axl     Erbb3      Mitf       MPS
cell
```

(continues on next page)

(continued from previous page)

```
C15_1   C15        #B9D7ED  6.328047  0.000000  0.000000 -0.727720
C15_2   C15        #B9D7ED  6.978424  3.604071  4.066950  0.170112
C15_3   C15        #B9D7ED  7.418106  5.479295  5.460087 -1.207896
C15_4   C15        #B9D7ED  8.461807  4.725196  2.711495 -2.571793
C15_5   C15        #B9D7ED  6.884476  6.314334  0.000000 -0.620660
...     ...            ...       ...       ...       ...       ...
C1_7     C1        #FF0000  7.931919  7.021924  3.656496  1.273881
C11_4   C11        #FF00AA  7.707152  6.642990  0.000000  0.644507
C11_5   C11        #FF00AA  7.078204  6.662490  0.000000  1.457377
C11_8   C11        #FF00AA  7.842476  4.391630  4.125155 -0.198301
C1_8     C1        #FF0000  8.679480  6.240505  0.000000 -0.234657

[83 rows x 6 columns]
```

Here is the information about the mutations:

[5]:
```
adata.var
```

[5]:
```
             CHROM        POS REF    ALT      START        END Allele  \
mutation
mutation_1    chr1   15815968   A  ['G']   15815968   15815968      G
mutation_2    chr1   37396158   G  ['A']   37396158   37396158      A
mutation_3    chr1   38045805   T  ['C']   38045805   38045805      C
mutation_4    chr1   51071476   G  ['A']   51071476   51071476      A
mutation_5    chr1   54997173   A  ['G']   54997173   54997173      G
...            ...        ...  ..    ...        ...        ...    ...
mutation_448  chrX  105877558   A  ['G']  105877558  105877558      G
mutation_449  chrX  134605536   A  ['G']  134605536  134605536      G
mutation_450  chrX  155214105   A  ['T']  155214105  155214105      T
mutation_451  chrX  155574460   A  ['G']  155574460  155574460      G
mutation_452  chrX  162761681   G  ['C']  162761681  162761681      C

                    Annotation Gene_Name Transcript_BioType     HGVS.c  \
mutation
mutation_1      missense_variant     Terf1     protein_coding   c.581A>G
mutation_2    synonymous_variant    Inpp4a     protein_coding  c.2622G>A
mutation_3      missense_variant     Eif5b     protein_coding  c.2732T>C
mutation_4      missense_variant    Tmeff2     protein_coding   c.448G>A
mutation_5      missense_variant     Sf3b1     protein_coding  c.2740T>C
...                          ...       ...               ...        ...
mutation_448  synonymous_variant      Atrx     protein_coding   c.675T>C
mutation_449    missense_variant   Hnrnph2     protein_coding   c.629A>G
mutation_450    missense_variant      Sat1     protein_coding   c.232T>A
mutation_451    missense_variant    Ptchd1     protein_coding  c.1748T>C
mutation_452    missense_variant     Rbbp7     protein_coding   c.123G>C

                   HGVS.p
mutation
mutation_1    p.Tyr194Cys
mutation_2    p.Val874Val
mutation_3    p.Val911Ala
mutation_4    p.Gly150Ser
mutation_5    p.Phe914Leu
```

(continues on next page)

(continued from previous page)

```
...                   ...
mutation_448  p.Gly225Gly
mutation_449  p.Tyr210Cys
mutation_450   p.Tyr78Asn
mutation_451  p.Val583Ala
mutation_452   p.Trp41Cys

[452 rows x 12 columns]
```

Now we will do some filteration to remove artifacts.

[6]:
```
tsc.pp.filter_mut_vaf_greater_than_coverage_mutant_greater_than(
    adata, min_vaf=0.4, min_coverage_mutant=20, min_cells=2
)
tsc.pp.filter_mut_reference_must_present_in_at_least(adata, min_cells=1)
tsc.pp.filter_mut_mutant_must_present_in_at_least(adata, min_cells=2)
```

```
Matrix with n_obs × n_vars = 83 × 268
Matrix with n_obs × n_vars = 83 × 267
Matrix with n_obs × n_vars = 83 × 267
```

[7]:
```
tsc.pp.build_scmatrix(adata)
df_in = adata.to_df()
```

[8]:
```
# df_out = tsc.tl.booster(
#     df_in,
#     alpha=0.001,
#     beta=0.2,
#     solver="SCITE",
#     sample_on="muts",
#     sample_size=20,
#     n_samples=9000,
#     n_jobs=16,
# )
df_out = tsc.tl.scistree(df_in, alpha=0.001, beta=0.2)
```

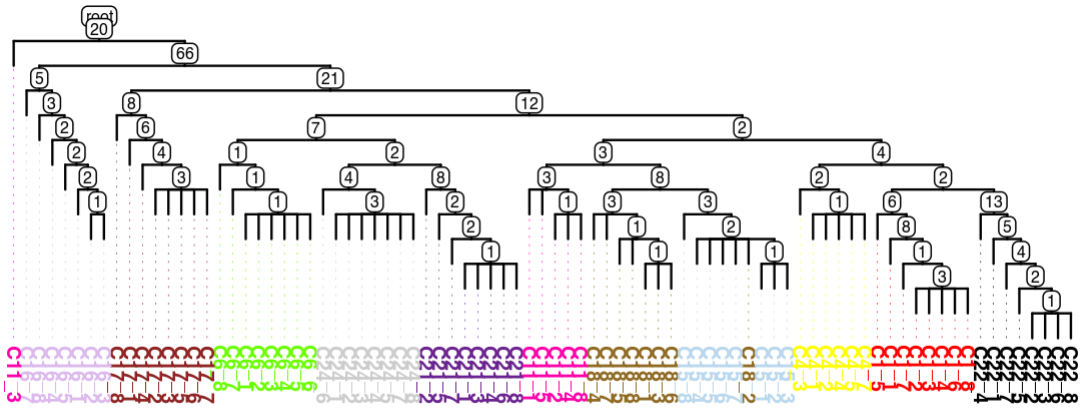
```
running ScisTree with alpha=0.001, beta=0.2
input -- size: 83x267
input -- 0: 9968#, 45.0%
input -- 1: 4020#, 18.1%
input -- NA: 8173#, 36.9%
input -- CF: False
output -- size: 83x267
output -- 0: 11308#, 51.0%
output -- 1: 10853#, 49.0%
output -- NA: 0#, 0.0%
output -- CF: True
output -- time: 59.0s (0:00:59.043201)
flips -- #0->1: 1881
flips -- #1->0: 27
flips -- #NA->0: 3194
flips -- #NA->1: 4979
rates -- FN: 0.320
```

(continues on next page)

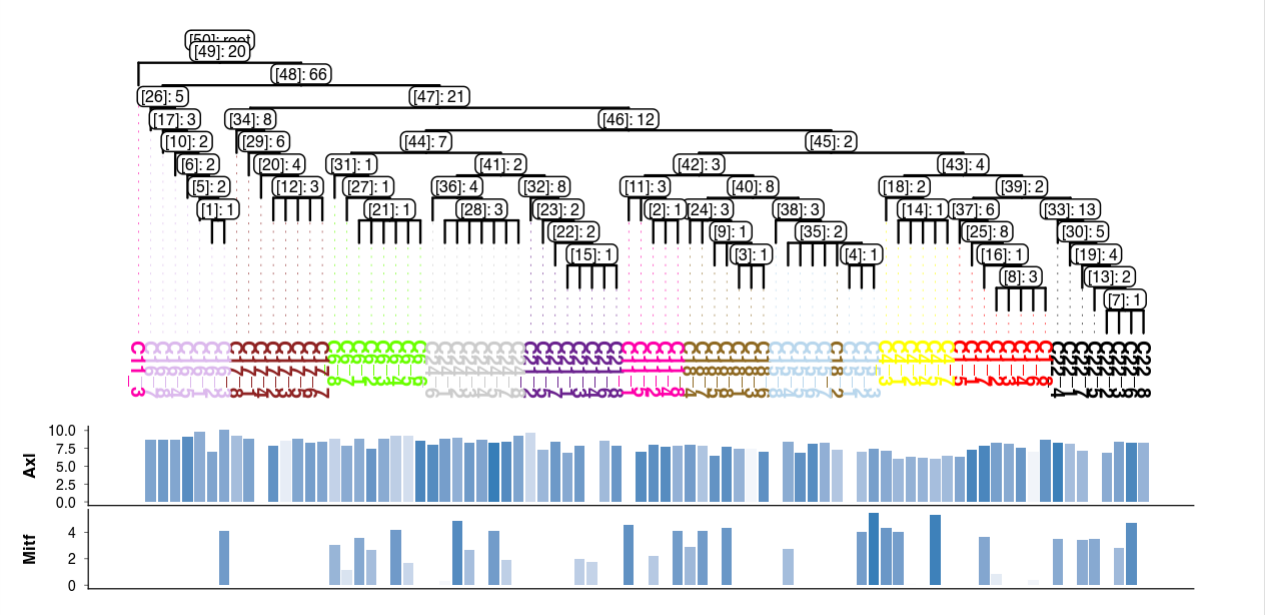
(continued from previous page)

```
rates -- FP: 0.00332758
rates -- NA: 0.369
score -- NLL: 4112.965352416734
```

[9]:
```
tree = tsc.ul.to_tree(df_out)
tsc.pl.dendro_tree(
    tree,
    cell_info=adata.obs,
    label_color="subclone_color",
    width=1200,
    height=500,
    dpi=200,
)
```



[10]:
```
tsc.pl.dendro_tree(
    tree,
    cell_info=adata.obs,
    label_color="subclone_color",
    width=1200,
    height=600,
    dpi=200,
    distance_labels_to_bottom=3,
    inner_node_type="both",
    inner_node_size=2,
    annotation=[
        ("bar", "Axl", "Erbb3", 0.2),
        ("bar", "Mitf", "Mitf", 0.2),
    ],
)
```



List of mutations branching at node with id [43]

[11]:
```
mut_ids = tree.graph['mutation_list'][tree.graph['mutation_list']['node_id'] == '[43]']
adata.var.loc[mut_ids.index]
```

[11]:
```
              CHROM        POS REF     ALT      START        END Allele  \
index
mutation_162   chr7   28042135   A  ['C']   28042135   28042135      C
mutation_349  chr13  103753116   T  ['G']  103753116  103753116      G
mutation_429  chr19    4035556   G  ['C']    4035556    4035556      C
mutation_8     chr1   74287097   T  ['G']   74287097   74287097      G

                    Annotation Gene_Name Transcript_BioType     HGVS.c  \
index
mutation_162    missense_variant     Psmc4     protein_coding  c.1109T>G
mutation_349  synonymous_variant     Srek1     protein_coding  c.1039A>C
mutation_429    missense_variant     Gstp1     protein_coding   c.550C>G
mutation_8      missense_variant      Pnkd     protein_coding   c.242T>G

                  HGVS.p
index
mutation_162  p.Ile370Ser
mutation_349  p.Arg347Arg
mutation_429  p.Leu184Val
mutation_8     p.Ile81Ser
```

[ ]:

1.9 Examples

This section contains various code snippets that demonstrate *trisicell*'s usage.

1.9.1 Reconstruction

Below is a gallery of examples for *trisicell.tl*.

Reconstruction

Below is a gallery of examples for *trisicell.tl*.

Construct lienage tree using Trisicell-Boost

This example shows how to construct a lineage tree using Trisicell-Boost on a binary single-cell genotype matrix.

```
import trisicell as tsc

# sphinx_gallery_thumbnail_path = "_static/thumbnails/trisicell-boost.png"
```

First, we load a binary test single-cell genotype data.

```
df_in = tsc.datasets.test()
df_in.head()
```

Next, using *trisicell.tl.booster()* we remove the single-cell noises from the input.

```
df_out = tsc.tl.booster(
    df_in,
    alpha=0.0000001,
    beta=0.1,
    solver="PhISCS",
    sample_on="muts",
    sample_size=15,
    n_samples=88,
    begin_index=0,
    n_jobs=1,
    dep_weight=50,
)
df_out.head()
```

```
SUBSAMPLING    (1/3):   0%|                                                    | 0/88 [00:
↪00<?, ?it/s]
SUBSAMPLING    (1/3):   0%|                                                    | 0/88 [00:
↪03<?, ?it/s]

DEPENDENCIES   (2/3):   0%|                                                    | 0/88 [00:
↪00<?, ?it/s]
DEPENDENCIES   (2/3):  88%|###################################     | 77/88 [00:00<00:00,␣
↪765.62it/s]
```

(continues on next page)

(continued from previous page)

```
DEPENDENCIES   (2/3): 100%|########################################| 88/88 [00:00<00:00,␣
↪765.13it/s]

RECONSTRUCTION (3/3):   0%|                                                | 0/20 [00:
↪00<?, ?it/s]
RECONSTRUCTION (3/3): 100%|########################################| 20/20 [00:00<00:00,␣
↪11781.75it/s]
input -- size: 20x20
input -- 0: 226#, 56.5%
input -- 1: 94#, 23.5%
input -- NA: 80#, 20.0%
input -- CF: False
output -- size: 20x20
output -- 0: 278#, 69.5%
output -- 1: 122#, 30.5%
output -- NA: 0#, 0.0%
output -- CF: True
output -- time: 3.4s (0:00:03.419101)
flips -- #0->1: 10
flips -- #1->0: 0
flips -- #NA->0: 62
flips -- #NA->1: 18
rates -- FN: 0.096
rates -- FP: 0.00000000
rates -- NA: 0.200
score -- NLL: 32.92976100177747
TREE_SCORE: 179.1110811296527
```

Finally, using `trisicell.ul.is_conflict_free_gusfield()` we check whether the inferred genotype matrix is conflict-free or not.

```
is_cf = tsc.ul.is_conflict_free_gusfield(df_out)
is_cf
```

```
True
```

Total running time of the script: (0 minutes 3.849 seconds)

Estimated memory usage: 15 MB

PYTHON MODULE INDEX

t

INDEX

Symbols

A

B

C

D

E

F