Allele Expression Consolidator

Release 1.0

Nicolás López, Jorge Finke, and Camilo Rocha

Jun 27, 2023

CONTENTS:

CHAPTER

ONE

ALLELECONSOLIDATOR

1.1 alleleconsolidator module

1.1.1 Algorithmic description

The procedure is explained in detail on the following paper, whose datasets can be found in the corresponding repository:

López-Rozo, N.; Ramirez-Castrillon, M.; Romero, M.; Finke, J.; Rocha, C. *Gene Expression Datasets for Two Versions of the Saccharum spontaneum AP85-441 Genome*. Data 2023, 8, 1. https://doi.org/10.3390/data8010001

Based on the output of BLASTN, the associations among the alleles in v2018 and v2019 are found to have repetitions. In the case of the mappings between v2018 to v2019, a CDS in the source could be associated with several CDS in the target. To generate a reasonable coverage, both mappings are combined by modeling the problem as a graph flow optimization problem [20] with multiple sources (v2018 alleles) and multiple targets (v2019 alleles).

A min-cost max-flow problem requires to compute a graph-matching (i.e., match at the level of nodes/vertices) with maximal cardinality (i.e., maximal number of connections), thus ensuring a maximal covering of the source-target associations. If more than one maximal matching is possible, then the cost of producing that maximal flow is to be minimized. In this case, identity scores can be considered to identify the matching with the greatest sum of identity scores, while still ensuring that a v2018 allele expression is used at most once. Since the algorithm implemented in networkx minimizes cost, the artificial cost fed to the min-cost max-flow algorithm is *pident* (i.e., percent identity) on each possible association between the two versions of the alleles.

1.1.2 API Reference

alleleconsolidator.**allele_transform**(*mapping_1*, *mapping_2*, *expression_1*, *result_folder='./'*, *expression_2='expression2.csv'*, *prefix_restricted='restricted'*, *blastn_titles='qseqid,sseqid,pident,length,mismatch,gapopen,qstart,qend,sstart,send,eva*
right_subset='qseqid', *verbose=False*)

This function transforms the expression values of the alleles in v2018 to the alleles in v2019. The transformation is done by solving a min-cost max-flow problem with multiple sources (v2018 alleles) and multiple targets (v2019 alleles).

Keep in mind that the function stores additional files in the result folder:

- *expression_2*: The mapped expression file for v2019, as CSV format.
- *prefix_restricted.pk*: The gene assignment, as a dictionary.

- *prefix_restricted.txt*: The gene assignment, as a table.

Parameters

- **mapping_1** (*str*) – The path to the mapping file from v2018 to v2019.
- **mapping_2** (*str*) – The path to the mapping file from v2019 to v2018.
- **expression_1** (*str*) – The path to the expression file of v2018.
- **result_folder** (*str*) – The path to the folder where the results will be saved.
- **expression_2** (*str*) – The name of the expression file of v2019.
- **prefix_restricted** (*str*) – The prefix of the restricted alleles.
- **blastn_titles** (*str*) – The titles of the BLASTN output.
- **right_subset** (*str*) – The subset identifier of the left nodes of the bipartite graph.
- **verbose** (*bool*) – If set to True, the function prints the time at the beginning and the end of the execution.

Return bipartite
The bipartite graph used for computing the min-cost max-flow.

Rtype bipartite
networkx.Graph

alleleconsolidator.**create_bipartite_graph**(*df_map*, *query_col='qseqid'*, *subject_col='sseqid'*, *weight_col='pident'*, *base_graph=None*)

This function creates a bipartite graph from the mapping dataframe, which was extracted from the BLASTN output. **Note**: The weight of the edges is multiplied by -1000 and truncated to be used as a cost in the min-cost max-flow algorithm.

Parameters

- **df_map** (*pandas.DataFrame*) – The mapping dataframe.
- **query_col** (*str*) – The name of the column containing the query sequences.
- **subject_col** (*str*) – The name of the column containing the subject sequences.
- **weight_col** (*str*) – The name of the column containing the weight of the edges.
- **base_graph** (*networkx.Graph*) – The base graph to be used. if set to None (default), a new graph is created.

Return bipartite
The bipartite graph.

Rtype bipartite
networkx.Graph

alleleconsolidator.**draw_multigraph**(*graph*, *subset_feat*, *x_offset=1*, *y_offset=1*, *subset_color=('green', 'blue', 'purple', 'red', 'orange', 'yellow')*, *savefig=False*, *filename='graph.pdf'*, *node_size=100*)

This function draws a multipartite graph. **Note**: The graph is drawn in the order of the subsets in the list of subset features. The nodes are

Parameters

- **graph** (*networkx.Graph*) – The graph to be drawn.

- **subset_feat** – The subset feature names. The order of the names is the order the subsets will be drawn.
- **x_offset** (*int*) – The x offset for the subsets and the drawn nodes.
- **y_offset** (*int*) – The y offset for the subsets and the drawn nodes.
- **subset_color** (*list[str]*) – The colors of the subsets.
- **savefig** (*bool*) – Whether to save the figure or not.
- **filename** (*str*) – The name of the file to save the figure.
- **node_size** (*int*) – The size of the nodes.

alleleconsolidator.**read_expression**(*file*, *sep=None*)

This function reads the expression file and returns a list of names and a matrix of expression values.

Parameters

- **file** (*str*) – The path to the expression file.
- **sep** (*str*) – The separator used in the file.

Return names
The list of names.

Rtype names
list

Return mat
The matrix of expression values.

Rtype mat
list[numpy.array]

Return headers
The headers of the table ("gene"/"allele" + accession names).

Rtype headers
str

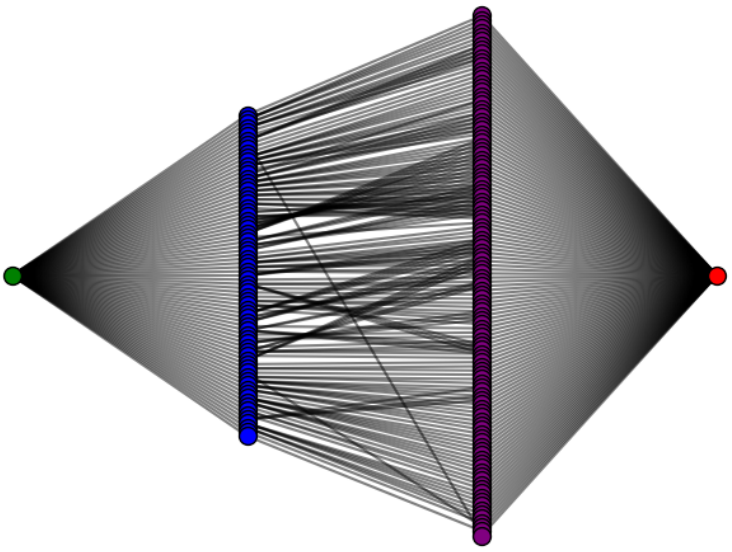
1.2 test module

1.2.1 Usage example:

The module *test.py* is available for showing the main functionalities of the alleleconsolidator module. To run it, use:

```
python3 test.py
```

It will compute the routines and generate plots for the example dataset, which is a subset of the data used in the reference paper.



This is the test file of the project. It loads the example files (from data folder) and runs the algorithm on them, storing the results in the test_results folder.

`test.main()`

This is the main function of the test file. It loads the example files (from data folder) and runs the algorithm on them, storing the results in the test_results folder.

CHAPTER

TWO

TEST AND USAGE

To test the package, run the following command in the root directory:

```
python3 test.py
```

It can also be tested with the following Python 3 script:

```
from alleleconsolidator import *

file_mapping_1 = "blast_1.txt"
file_mapping_2 = "blast_2.txt"
file_expression_in = "expression.csv"
file_expression_out = "expression_2.csv"
folder_in = "data/"
folder_out = "test_results/"
prefix_restricted = "restricted"
g = allele_transform(folder_in + file_mapping_1, folder_in + file_mapping_2,
                     folder_in + file_expression_in, folder_out,
                     file_expression_out, prefix_restricted, verbose=True)
print("Results from the allele transform algorithm (first 20 lines):")
with open(f"{folder_out}{prefix_restricted}.txt", "r") as f:
    for line in f.readlines()[:20]:
        print(line.strip())

# Plotting the resulting graph
draw_multigraph(g, ["SS", "qseqid", "sseqid", "ST"], x_offset=10,
                savefig=True, filename="test_results/graph.pdf")
```

PYTHON MODULE INDEX

INDEX