Delab Trees

A python library to analyze conversation trees

Julian Dehne

2024-11-18

At a glance

By the end of this tutorial, you will be able to

- Analyze the integrity of the social media conversation
- Use network analysis to extract longer reply path that might represent actual deliberation
- Use network analysis to show which author is the most central in the discussion

Table of Content

Introduction

Description

- This notebook introduces the python library delab_trees and showcases on some examples how it can be useful in dealing with social media data.

Target Audience

- This library is intended for advanced CSS researchers that have a solid background in network computing and python
- Motivated intermediate learners may use some of the toolings as a blackbox to arrive at the conversation pathways later used in their research

Prerequisites

Before you begin, you need to know the following technologies.

- python
- networkX
- pandas

Set-up

- In order to run this tutorial, you need at least Python >= 3.9
- the library will install all its dependencies, just run

```
pip install delab_trees
```

Social Science Usecases

This learning resource is useful if you have encountered one of these three use cases:

- deleted posts in your social media data
- interest in author interactions on social media
- huge numbers of conversation trees (scalability)
- discussion mining (finding actual argumentation sequences in social media)

Sample Input and Output Data

Example data for Reddit and Twitter are available here https://github.com/juliandehne/delab-trees/raw/main/delab_trees/data/dataset_[reddit|twitter]_no_text.pkl. The data is structure only. Ids, text, links, or other information that would break confidentiality of the academic access have been omitted.

The trees are loaded from tables like this:

	tree_id	post_id	parent_id	author_id	text	created_at
0	1	1	nan	james	I am James	2017-01-01 01:00:00
1	1	2	1	mark	I am Mark	2017-01-01 02:00:00
2	1	3	2	steven	I am Steven	2017-01-01 03:00:00
3	1	4	1	john	I am John	2017-01-01 04:00:00
4	2	1	nan	james	I am James	2017-01-01 01:00:00
5	2	2	1	mark	I am Mark	2017-01-01 02:00:00
6	2	3	2	steven	I am Steven	2017-01-01 03:00:00
7	2	4	3	john	I am John	2017-01-01 04:00:00

This dataset contains two conversational trees with four posts each.

Currently, you need to import conversational tables as a pandas dataframe like this:

```
import os
import sys
import warnings
import numpy as np  # Example module that might trigger the warning

# assert that you have the correct environment
print(f"Active conda environment: {os.getenv('CONDA_DEFAULT_ENV')}")

# assert that you have the correct python version (3.9)
print(f"Python version: {sys.version}")

# Suppress the specific VisibleDeprecationWarning
warnings.filterwarnings("ignore", category=np.VisibleDeprecationWarning)
```

```
# the interesting code
from delab_trees import TreeManager
import pandas as pd


d = {'tree_id': [1] * 4,
     'post_id': [1, 2, 3, 4],
     'parent_id': [None, 1, 2, 1],
     'author_id': ["james", "mark", "steven", "john"],
     'text': ["I am James", "I am Mark", " I am Steven", "I am John"],
     "created_at": [pd.Timestamp('2017-01-01T01'),
                    pd.Timestamp('2017-01-01T02'),
                    pd.Timestamp('2017-01-01T03'),
                    pd.Timestamp('2017-01-01T04')]}
df = pd.DataFrame(data=d)
manager = TreeManager(df)
# creates one tree
test_tree = manager.random()
test_tree
```

```
Active conda environment: notebook
Python version: 3.9.19 | packaged by conda-forge | (main, Mar 20 2024, 12:50:21)
[GCC 12.3.0]
loading data into manager and converting table into trees...

2024-12-18 09:55:04.857070: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not find cuda dri
2024-12-18 09:55:04.899633: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not find cuda dri
2024-12-18 09:55:04.900762: I tensorflow/core/platform/cpu_feature_guard.cc:182] This Tensor
critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with
2024-12-18 09:55:06.978601: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-
TRT Warning: Could not find TensorRT
  0%|          | 0/1 [00:00<?, ?it/s]100%|██████████| 1/1 [00:00<00:00, 142.77it/s]

<delab_trees.delab_tree.DelabTree at 0x7fcd381ee790>
```

Note that the tree structure is based on the parent_id matching another rows post_id.

You can now analyze the reply trees basic metrics:

```
from delab_trees.test_data_manager import get_test_tree
from delab_trees.delab_tree import DelabTree
import warnings
import numpy as np

# Suppress only VisibleDeprecationWarning
warnings.filterwarnings("ignore", category=np.VisibleDeprecationWarning)

test_tree : DelabTree = get_test_tree()
assert test_tree.average_branching_factor() > 0

print("number of posts in the conversation: ", test_tree.total_number_of_posts())
```

```
loading data into manager and converting table into trees...
number of posts in the conversation:  4
```

```
  0%|          | 0/1 [00:00<?, ?it/s]100%|██████████| 1/1 [00:00<00:00, 106.93it/s]
```

Tool application

Use Case 1: Analyze the integrity of the social media conversation

For this we use the provided anonymized sample data (which is real, still):

```
from delab_trees.test_data_manager import get_test_manager

manager = get_test_manager()
manager.describe()
```

```
loading data into manager and converting table into trees...
```

```
  0%|          | 0/6 [00:00<?, ?it/s]100%|██████████| 6/6 [00:00<00:00, 565.41it/s]
```

```
'The dataset contains 6 conversations and 24 posts in total.\nThe average depth of the longe
```

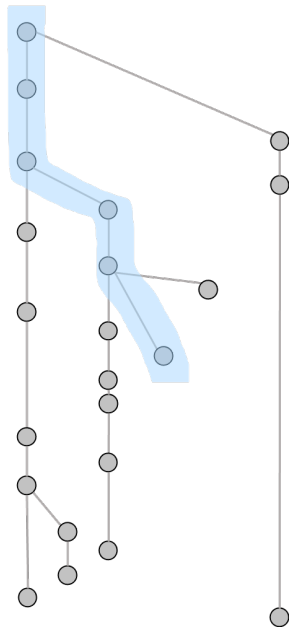
In order to check if all the conversations are valid trees which in social media data, they often are not, simply call:

```
manager.validate(break_on_invalid=False, verbose=False)
```

```
  0%|          | 0/6 [00:00<?, ?it/s] 83%|████████▎ | 5/6 [00:00<00:00, 1703.62it/s]
```

```
False
```

Use Case 2: Extract Pathways



As an analogy with offline-conversations, we are interested in longer reply-chains as depicted in Figure 1. Here, the nodes are the posts, and the edges read from top to bottom as a post answering another post. The root of the tree is the original post in the online conversation. Every online forum and social media thread can be modeled this way because every post except the root post has a parent, which is the mathematical definition of a recursive tree structure.

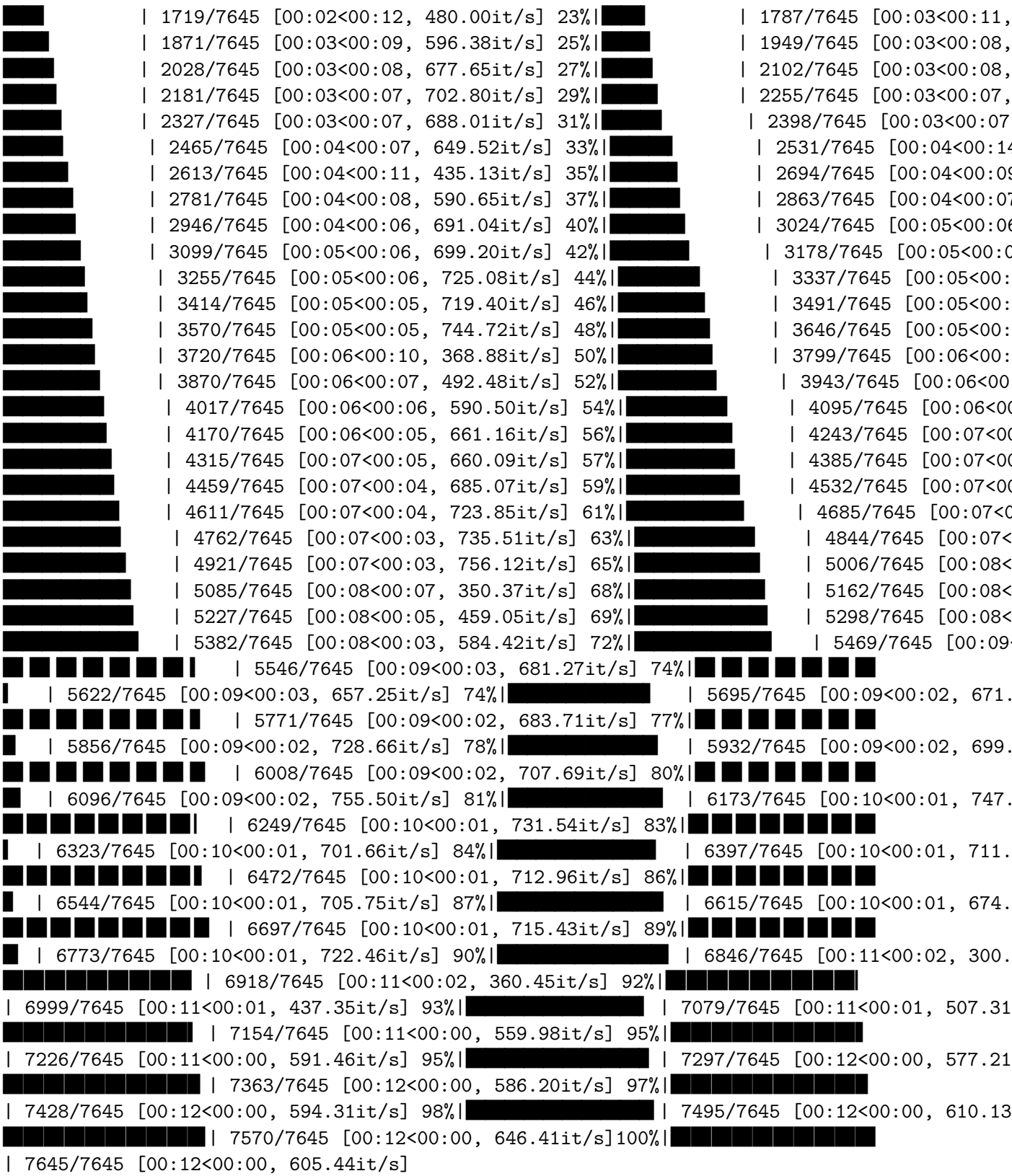
The marked path is one of many pathways that can be written down like a transcript from a group discussion. Pathways can be defined as all the paths in a tree that start with the root and end in a leaf (a node without children). This approach serves the function of filtering linear reply-chains in social media (see Wang, Joshi, and Cohen (2008); Nishi et al. (2016)), that can be considered an online equivalent of real-life discussions.

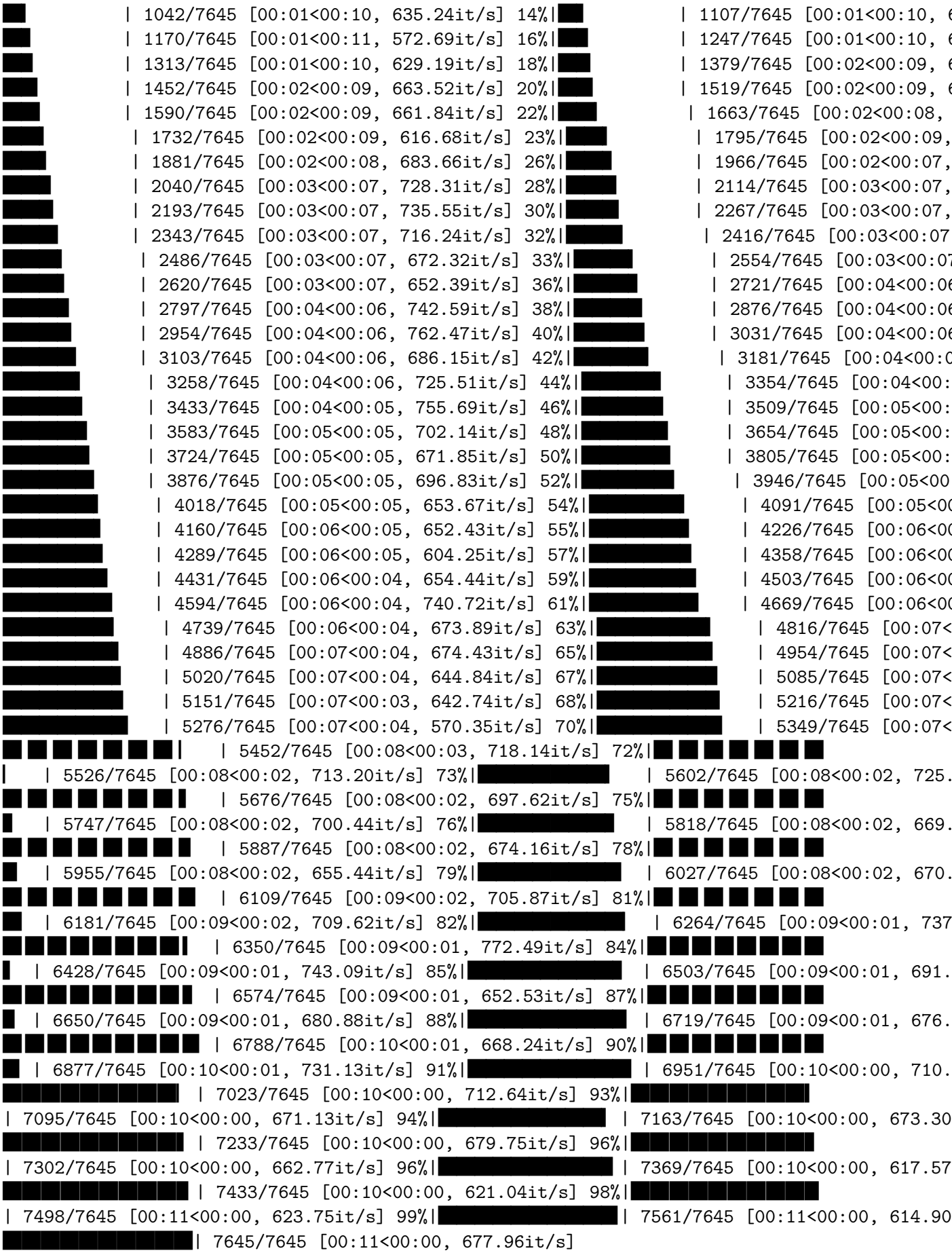
In order to have a larger dataset available we are going to load the provided dataset and run the flow_computation for each tree.

```
# get the sample trees
from delab_trees.test_data_manager import get_social_media_trees

social_media_tree_manager = get_social_media_trees()

# compute the flows
flow_list = [] # initialize an empty list
tree: DelabTree = None

for tree_id, tree in social_media_tree_manager.trees.items():
    flows = tree.get_conversation_flows(as_list=True)
    flow_list.append(flows)

print(len(flow_list), " were found")

# now we are only interested in flows of length 5 or more

# Filter to only include lists with length 5 or more
filtered_lists = [lst for lst in flow_list if len(lst) >= 7]

print(len(filtered_lists), " lists with length > 7 were found")
```

```
loading data into manager and converting table into trees...
6235  were found
5218  lists with length > 7 were found
```

```
  0%|          | 0/7645 [00:00<?, ?it/s]  1%|          | 64/7645 [00:00<00:12, 622.68it/s]
▏          | 139/7645 [00:00<00:10, 691.56it/s]  3%|▎          | 213/7645 [00:00<00:10, 710.
▏          | 286/7645 [00:00<00:10, 714.51it/s]  5%|▍          | 358/7645 [00:00<00:10, 707.
▏          | 429/7645 [00:00<00:10, 677.39it/s]  7%|▋          | 500/7645 [00:00<00:10, 687.
▏          | 569/7645 [00:00<00:10, 680.71it/s]  8%|▊          | 638/7645 [00:00<00:10, 680.
▏          | 708/7645 [00:01<00:10, 685.10it/s] 10%|▉          | 780/7645 [00:01<00:09, 693.
▏          | 851/7645 [00:01<00:09, 697.09it/s] 12%|█          | 922/7645 [00:01<00:17, 393
▏          | 992/7645 [00:01<00:14, 452.20it/s] 14%|█▏         | 1064/7645 [00:01<00:12, 50
▏          | 1128/7645 [00:01<00:12, 533.02it/s] 16%|█▍         | 1203/7645 [00:01<00:10, 5
▏          | 1282/7645 [00:02<00:10, 634.24it/s] 18%|█▌         | 1352/7645 [00:02<00:09, 6
▏          | 1423/7645 [00:02<00:09, 661.97it/s] 20%|█▋         | 1496/7645 [00:02<00:09, 6
▏          | 1572/7645 [00:02<00:08, 702.05it/s] 22%|█▊         | 1644/7645 [00:02<00:14,
```


| 1719/7645 [00:02<00:12, 480.00it/s] 23%| | 1787/7645 [00:03<00:11,
| 1871/7645 [00:03<00:09, 596.38it/s] 25%| | 1949/7645 [00:03<00:08,
| 2028/7645 [00:03<00:08, 677.65it/s] 27%| | 2102/7645 [00:03<00:08,
| 2181/7645 [00:03<00:07, 702.80it/s] 29%| | 2255/7645 [00:03<00:07,
| 2327/7645 [00:03<00:07, 688.01it/s] 31%| | 2398/7645 [00:03<00:07
| 2465/7645 [00:04<00:07, 649.52it/s] 33%| | 2531/7645 [00:04<00:14
| 2613/7645 [00:04<00:11, 435.13it/s] 35%| | 2694/7645 [00:04<00:09
| 2781/7645 [00:04<00:08, 590.65it/s] 37%| | 2863/7645 [00:04<00:07
| 2946/7645 [00:04<00:06, 691.04it/s] 40%| | 3024/7645 [00:05<00:06
| 3099/7645 [00:05<00:06, 699.20it/s] 42%| | 3178/7645 [00:05<00:0
| 3255/7645 [00:05<00:06, 725.08it/s] 44%| | 3337/7645 [00:05<00:
| 3414/7645 [00:05<00:05, 719.40it/s] 46%| | 3491/7645 [00:05<00:
| 3570/7645 [00:05<00:05, 744.72it/s] 48%| | 3646/7645 [00:05<00:
| 3720/7645 [00:06<00:10, 368.88it/s] 50%| | 3799/7645 [00:06<00:
| 3870/7645 [00:06<00:07, 492.48it/s] 52%| | 3943/7645 [00:06<00
| 4017/7645 [00:06<00:06, 590.50it/s] 54%| | 4095/7645 [00:06<00
| 4170/7645 [00:06<00:05, 661.16it/s] 56%| | 4243/7645 [00:07<00
| 4315/7645 [00:07<00:05, 660.09it/s] 57%| | 4385/7645 [00:07<00
| 4459/7645 [00:07<00:04, 685.07it/s] 59%| | 4532/7645 [00:07<00
| 4611/7645 [00:07<00:04, 723.85it/s] 61%| | 4685/7645 [00:07<0
| 4762/7645 [00:07<00:03, 735.51it/s] 63%| | 4844/7645 [00:07<
| 4921/7645 [00:07<00:03, 756.12it/s] 65%| | 5006/7645 [00:08<
| 5085/7645 [00:08<00:07, 350.37it/s] 68%| | 5162/7645 [00:08<
| 5227/7645 [00:08<00:05, 459.05it/s] 69%| | 5298/7645 [00:08<
| 5382/7645 [00:08<00:03, 584.42it/s] 72%| | 5469/7645 [00:09
| 5546/7645 [00:09<00:03, 681.27it/s] 74%|
| 5622/7645 [00:09<00:03, 657.25it/s] 74%| | 5695/7645 [00:09<00:02, 671.
| 5771/7645 [00:09<00:02, 683.71it/s] 77%|
| 5856/7645 [00:09<00:02, 728.66it/s] 78%| | 5932/7645 [00:09<00:02, 699.
| 6008/7645 [00:09<00:02, 707.69it/s] 80%|
| 6096/7645 [00:09<00:02, 755.50it/s] 81%| | 6173/7645 [00:10<00:01, 747.
| 6249/7645 [00:10<00:01, 731.54it/s] 83%|
| 6323/7645 [00:10<00:01, 701.66it/s] 84%| | 6397/7645 [00:10<00:01, 711.
| 6472/7645 [00:10<00:01, 712.96it/s] 86%|
| 6544/7645 [00:10<00:01, 705.75it/s] 87%| | 6615/7645 [00:10<00:01, 674.
| 6697/7645 [00:10<00:01, 715.43it/s] 89%|
| 6773/7645 [00:10<00:01, 722.46it/s] 90%| | 6846/7645 [00:11<00:02, 300.
| 6918/7645 [00:11<00:02, 360.45it/s] 92%|
| 6999/7645 [00:11<00:01, 437.35it/s] 93%| | 7079/7645 [00:11<00:01, 507.31
| 7154/7645 [00:11<00:00, 559.98it/s] 95%|
| 7226/7645 [00:11<00:00, 591.46it/s] 95%| | 7297/7645 [00:12<00:00, 577.21
| 7363/7645 [00:12<00:00, 586.20it/s] 97%|
| 7428/7645 [00:12<00:00, 594.31it/s] 98%| | 7495/7645 [00:12<00:00, 610.13
| 7570/7645 [00:12<00:00, 646.41it/s]100%|
| 7645/7645 [00:12<00:00, 605.44it/s]
0%| | 0/7645 [00:00<?, ?it/s] 1%| | 77/7645 [00:00<00:10, 742.97it/s]
| 153/7645 [00:00<00:10, 743.23it/s] 3%| | 228/7645 [00:00<00:11, 658.
| 295/7645 [00:00<00:11, 640.35it/s] 5%| | 367/7645 [00:00<00:10, 664.
| 434/7645 [00:00<00:10, 662.21it/s] 7%| | 504/7645 [00:00<00:10, 673.
| 596/7645 [00:00<00:09, 731.99it/s] 9%| | 670/7645 [00:00<00:09, 726.
| 759/7645 [00:01<00:08, 773.43it/s] 11%| | 837/7645 [00:01<00:10, 664.
| 907/7645 [00:01<00:10, 665.06it/s] 13%| | 976/7645 [00:01<00:10, 637



Use Case 3: compute the centrality of authors in the conversation

```
test_tree : DelabTree = get_test_tree()
metrics =  test_tree.get_author_metrics() # returns a map with author ids as keys
for author_id, metrics in metrics.items():
    print("centrality of author {} is {}".format(author_id, metrics.betweenness_centrality))
```

```
loading data into manager and converting table into trees...
centrality of author john is 0.0
centrality of author james is 0.0
centrality of author mark is 0.16666666666666666
centrality of author steven is 0.0

  0%|          | 0/1 [00:00<?, ?it/s]100%|██████████| 1/1 [00:00<00:00, 92.51it/s]
```

The result shows, that only mark is central in the sense that he is answered to and has answered. In bigger trees, this makes more sense.

Library Documentation

For an overview over the different functions, have a look here

Conclusion

Now you should be able to analyze social media trees effectively. For any questions, write me an email. I am happy to help!

Also I would be happy if someone is interested in doing research and writing a publication with this library!

Exercises or Challenges (Optional)

Learning exercises are forthcoming! But for now you should click on the binderhub link on the top to get a notebook in Jupyterlab, where you can play around with the code.

FAQs (Optional)

This will be filled if more people use the library!

Nishi, R., T. Takaguchi, K. Oka, T. Maehara, M. Toyoda, K.-i. Kawarabayashi, and N. Masuda. 2016. "Reply Trees in Twitter: Data Analysis and Branching Process Models." *Social Network Analysis and Mining* 6 (1): 26.

Wang, Y.-C., M. J. M. Joshi, and W. Cohen. 2008. "Recovering Implicit Thread Structure in Newsgroup Style Conversations." *Proceedings of the International AAAI Conference on Web and Social Media* 2 (1): 152–60.