Finding the Math Department's Deep Structure

Illinois Geometry Lab

Faculty Mentor: Yuliy Baryshnikov
Project Leaders: Haoyuan Li, Anji Dong, Ning Jiang
IGL Scholars: Prajeet Basu, Zifan Dong, Ziqi Xu, Adam Wawrowski, Qingyu Yi

University of Illinois at Urbana-Champaign

Midterm Presentation
October 15, 2023

Project Goals

Barycenters in the Space of (Phylogenetic) Trees:

- Utilize existing dataset to generate distance matrices derived from similarities between references, journals, MSC, keywords, extended coauthors networks.
- Use these distances to construct phylogenetic trees.
- Evaluate the resulting trees and implied clusterings for consistency with the existing department structure.
- Aggregate the trees generated in the previous steps to plausible barycenters.
- Use these barycenters to identify research clusters of the department.

Introduction

The main task of our team is to find Barycenters in the space of (Phylogenetic) Trees.

1. Raw Data: Generating raw data from paper and reference data for each professor, create similarity matrix.
2. Similarity Matrix: Parsing the Journals Data and References Data to create the corresponding similarity matrices
3. Distance Matrix: Based on the similarity matrix, we convert it into a distance matrix
4. Phylogenetic Tree: Using the phylogenetic trees, find reasonable median of them
5. Newick Format Tree: Using the Newick Format Tree, compute the geodesic distance

Raw Data

We have paper and reference data for each professor. Our first step is to sort out all the data and generate raw data where the column is the professor's name and the row is the paper or reference.


```
{"AuthorID": "615972", "Papers": {"MR4524115": {"Title": "Congruences like Atkin's for the partition
function.", "PaperID": "MR4524115", "Authors": [["Ahlgren, Scott", "615972"], ["Allen, Patrick B.",
"823416"], ["Tang, Shiang", "1189619"]], "Journal_Name": "Trans. Amer. Math. Soc. Ser. B",
"Publication_Year": "2022", "References": ["MR2000466", "MR1802511", "MR0268123", "MR0214540",
"MR1862931", "MR1874518", "MR0205958", "MR0227105", "MR1046750", "MR1605752", "MR2112196", "MR1176206",
"MR0205975", "MR1779182", "MR2988821", "MR0265287", "MR0092801", "MR1745012", "MR3066773", "MR1544457",
"MR1544457", "MR0819838", "MR0434996", "MR0885783", "MR2920749", "MR0332663", "MR3616493", "MR2251155",
"MR2446602", "MR1581588", "MR1829808", "MR3263525"], "Codes": "11F33,(11F80,11P83)"}, "MR4039543":
{"Title": "Dissections of strange q-series.", "PaperID": "MR4039543", "Authors": [["Ahlgren, Scott",
"615972"], ["Kim, Byungchan", "847992"], ["Lovejoy, Jeremy", "671259"]], "Journal_Name": ["Ann. Comb.",
"5510"], "Publication_Year": "2019", "References": ["MR3376226", "MR0399528", "MR2135178", "MR3435730",
"MR0434929", "MR3345301", "MR3375650", "MR2105705", "MR2267673", "MR1701924", "MR3376233", "MR1860536",
"MR2757599"], "Codes": "33D15,(05A30)"}, "MR3985121": {"Title": "Maass forms and the mock theta
function $f(q)$f(q).", "PaperID": "MR3985121", "Authors": [["Ahlgren, Scott", "615972"], ["Dunn,
Alexander", "1201899"]], "Journal_Name": ["Math. Ann.", "734"], "Publication_Year": "2019",
"References": ["MR2231957", "MR3801438", "arXiv:1801.08174", "MR3229965", "MR0200258", "MR2035813",
"MR2669637", "MR0066496", "MR1625181", "MR2431250", "MR2296066", "MR2435749", "MR3729259", "MR2231957",
"MR0049927", "MR1324141", "MR2926988", "MR0931205", "MR3242661", "MR2672303", "MR0439755", "MR0711197",
"MR0870736", "MR1937203", "MR0265287", "MR1745392", "MR1501943", "MR3458951", "MR2723248", "MR2023036",
"MR1575213", "MR0008618", "MR0364103", "MR0750670", "MR2641204", "MR0472707", "MR3857941", "MR1573993",
"MR0080122", "MR3635360", "MR2605321"], "Codes": "11F37,(11F30,11L05,11P82)"}, "MR3852563": {"Title":
"A polyharmonic Maass form of depth 3/2 for SL2(Z).", "PaperID": "MR3852563", "Authors": [["Ahlgren,
Scott", "615972"], ["Andersen, Nickolas", "997274"], ["Samart, Detchat", "1044112"]], "Journal_Name":
["J. Math. Anal. Appl.", "3591"], "Publication_Year": "2018", "References": ["MR0429750", "MR2776366",
"MR3353542", "MR3801423", "MR3500994", "MR3439692", "MR3354436", "MR3592591", "MR2456627", "MR2437679",
```

Raw Data

```
import json
import os
import glob
import re
import csv
path = os.getcwd()

def read_json_file(file):
    with open(file, 'r') as f:
        data = json.load(f)
    return data


def format_author_name(file_name):
    name_with_underscores = os.path.basename(file_name).replace('_papers.json', '')
    # Insert a space before each capital letter to format the name
    formatted_name = re.sub(r"(?<=\w)([A-Z])", r" \1", name_with_underscores)
    if formatted_name[-1].isupper():
        formatted_name += '.'
    return formatted_name

#def get_cite(data):


def get_data(data):

    #if data['AuthorID'] == author_id:
        author_id = data['AuthorID']
        # List of journals with duplicates removed
        #change paper['Journal_Name'][0] to paper['Journal_Name'][1] if you want journal IDs instead of names
        journal_names = [paper['Journal_Name'][0] for paper in data['Papers'].values() if paper['Journal_Name']]
        unique_journal_names = list(set(journal_names))
        print('Journals:', unique_journal_names)
        print("Number of journals:", len(unique_journal_names))

        # List of co_msc with duplicates removed
        msc_s = [msc for paper in data['Papers'].values() for msc in paper['Codes']] #change author[0] to author[1] if you want author IDs instead of names
        unique_msc_factor = list(set(msc_s))
        #print('Coauthors:', unique_co_authors)
        print("Number of Msc:", len(unique_msc_factor))

        # List of references with duplicates removed
        references = [reference for paper in data['Papers'].values() for reference in paper['References']]
        unique_references = list(set(references))
        #print('References:', unique_references)
        print("Number of refs", len(unique_references))

        # List of years published with duplicates removed
        publication_years = [paper['Publication_Year'] for paper in data['Papers'].values()]
        unique_publication_years = list(set(publication_years))
        print('Publication Years:', unique_publication_years)
        print("\n ")


with open(path + "/data/author_ids.json") as file:
    author_ids = json.load(file)


paper_files = glob.glob(path + '/data/papers/*_papers.json')
for file in paper_files:
    print(file)
    if file == path + '/data/papers/all_authors_papers.json':
       continue
    data = read_json_file(file)
    prof_name = format_author_name(file)
    print(" author name:", prof_name, "\n")
    get_data(data)
```

Raw Data

This is the raw data we generated.

The raw data for references:

Professors	References
Ahlgren Sc	['MR1671788', 'MR2134268', 'MR1544457', 'MR1184764', 'MR0819838', 'MR2155234', 'MR1849264', '
Albin Pierr	['MR2763625', 'MR0656119', 'MR2141470', 'MR2082390', 'MR0920965', 'MR0530173', 'MR1042214', '
Balogh Jó	['MR3843323', 'MR0710892', 'MR1710238', 'MR0144366', 'MR3338027', 'MR0151956', 'MR2791450', '
Baryshnik	['MR1855315', 'MR3086266', 'MR2749710', 'MR2152378', 'MR1115120', 'MR2145196', 'MR0243112', '
Berwick-E	['MR1758754', 'MR1953295', 'MR2125040', 'MR1869850', 'MR2439561', 'MR0092927', 'MR1701599', '
Boca Flori	['arXiv:1805.09312', 'MR1490645', 'MR0561974', 'MR3466857', 'MR0480290', 'MR1221111', 'MR2227(
Bradlow S	['MR2142332', 'MR1351164', 'MR1124279', 'MR0953674', 'MR1371207', 'MR1924201', 'arXiv:1111.255
Bronski Ja	['MR3214901', 'MR1108902', 'MR2152331', 'MR0284642', 'MR0475242', 'MR1783382', 'MR1235042', '

The raw data for journals:

Professors	Journals
Ahlgren Sc	['r', '5510', '734', '3591', '6356', '3003', '3544', '3003', '3544', '8769', '6116', '1992', '5143', None, '2807', '6316', '3544',
Albin Pierr	['n', '6598', '6356', '2267', '3544', '2869', '477', None, '6278', '6117', '6415', '6859', None, '6672', '5488', '2508', '6598',
Balogh Jó	['l', '2395', '2346', '3098', '3098', '3737', '1503', '6317', '615', '5169', '2346', None, '4168', '1985', '1503', '4309', '859', '
Baryshnik	['o', '1635', '9349', '5665', '4410', '4950', '9349', '6859', '2395', None, '5860', '2292', None, '6356', None, '450', '3544', N
Berwick-E	['e', '2183', '6274', '3544', '1470', '3548', '1529', None]
Boca Flori	['c', '3473', '5961', '3003', '3713', '421', '4315', '4315', '6116', '6364', '7979', '4265', '4675', '1589', None, '1818', '6116'
Bradlow S	['n', '4499', '449', None, '5741', '8395', '3555', '4135', '4348', '3098', '5497', '6277', '1743', None, '1141', '400', '3714', '3
Bronski Ja	['o', '450', '3473', '6038', '5283', '6038', '5170', '4178', '3473', '5158', '5763', '8621', '4178', '5158', None, '6750', '1529'
Cooney Da	['u', '6182', '2929', '1187', '6182', '2929', '815']
De Ville R	['h', '3473', '4178', '1985', '8456', '1985', '6038', '4178', '3473', '2929', '4178', '5158', '5763', '4872', '4872', '5961', '611
Dey Partha	['n', '6038', '1081', '1081', '1734', '2342', '4083', '6424', '1081', None, '1734', '5740']
Di Frances	['.', '269', '4633', None, '6301', '4633', '6038', '6038', '1529', None, '6301', None, '3691', '4913', '6301', '6301', '3548', '4

Similarity Matrix

Then we parse the Journals Data and References Data to create the corresponding similarity matrices.

1. Similarity Matrix for Journals S[i, j] = Sum of all papers (from prof i and prof j) written in the common journals of professor i and professor j. For example, if A published 3 papers in journal X and 4 papers in journal Y, while B published 2 papers in journal X and 5 papers in journal Y, then entry (A, B) of this dataset would be 3+2+4+5 = 14
2. Similarity Matrix for References S[i, j] = Number of times Prof i and Prof j use the same reference. For example, if A referred to paper x, y, z in her papers, and B referred to paper w, x, y in his papers, then entry (A, B) of this dataset would be 2.

Similarity Matrix

```
filename = r'C:\Users\praje\OneDrive\Documents\IGL_DEEP_STRUCTURES\CurrRawMatrices\prof_references_2023.csv'
out_filename = 'similarity_references.csv'
df1 = pd.read_csv(filename)
df1['Professors'] = df1['Professors'].str.strip()

columns_ = df1["Professors"]
#Need to change it from a series to a list to get rid of the header
cols = list(columns_)

# create empty matrix
data = np.zeros(shape = (len(columns_), len(columns_)), dtype=int)
out_df_refs = pd.DataFrame(data, columns =cols, index =cols)
```

Similarity Matrix

```
#Iterate through the professors
for i in range(len(columns_)):

    #First professor
    curr_prof1 = columns_[i]

    for j in range(len(columns_)):
        #Second professor being compared to the first
        curr_prof2 = columns_[j]

        #print('\r{}, {}'.format(curr1, c2), end = '')

        #Let us find the two lists of references
        ref_1 = df1.iat[i, 1].split(",")
        ref_2 = df1.iat[j, 1].split(",")

        #Total in common
        common_num = 0

        set_1 = set(ref_1)
        set_2 = set(ref_2)

        # Find the common elements using set intersection
        common_elements = set_1.intersection(set_2)

        # Get the count of common elements
        common_num = len(common_elements)

        out_df_refs.loc[curr_prof1, curr_prof2] = common_num
        out_df_refs.loc[curr_prof2, curr_prof1] = common_num

# save matrix to csv file
out_df_refs.to_csv(out_filename)
```

Similarity Matrix

This is the similarity matrix we generated.

Similarity Matrix for Journals

	Ahlgren Scott D.	Albin Pierre	Balogh József	Baryshnikov Yuliy M.	Berwick-Evans Daniel	Boca Florin-Petre	Bradlow Steven Benjamin	Bronski Jared C.	Cooney Daniel B.	De Ville R E Lee	...	Tolman Susan	Tumanov Alexander
Ahlgren Scott D.	56	14	22	30	2	35	19	12	0	4	...	20	26
Albin Pierre	14	26	5	20	2	13	10	11	0	2	...	11	14
Balogh József	22	5	177	26	0	10	12	3	0	5	...	3	13
Baryshnikov Yuliy M.	30	20	26	75	0	20	15	38	0	25	...	16	24
Berwick-Evans Daniel	2	2	0	0	8	0	0	0	0	0	...	8	0

5 rows × 70 columns

Similarity Matrix for References

```
out_df_refs.head()
```

Python

	Ahlgren Scott D.	Albin Pierre	Balogh József	Baryshnikov Yuliy M.	Berwick-Evans Daniel	Boca Florin-Petre	Bradlow Steven Benjamin	Bronski Jared C.	Cooney Daniel B.	De Ville R E Lee	...	Tolman Susan	Tumanov Alexander	Tyson Jeremy T.	Tzirakis Nikolaos	Wei Wei	Wu Xuan	Yong Alexander T F.	Young Amanda	Zaharescu Alexandru
Ahlgren Scott D.	475	0	0	0	2	7	0	0	0	0	...	1	0	0	0	0	1	1	0	72
Albin Pierre	0	601	0	10	7	0	4	1	0	1	...	9	2	7	0	0	0	0	0	3
Balogh József	0	0	1699	5	0	1	0	4	0	6	...	0	0	0	0	1	1	14	1	8
Baryshnikov Yuliy M.	0	10	5	607	1	2	3	2	0	9	...	4	7	12	1	0	1	10	1	5
Berwick-Evans Daniel	2	7	0	1	164	0	0	0	0	0	...	6	0	0	0	0	0	0	0	0

5 rows × 70 columns

Distance Matrix

Based on the similarity matrix, we convert it into a distance matrix.
Define the distance matrix using formula $distance = 1/(r^n)$, where n is the entries in similarity matrix and r is a parameter.

```
def d1(similarity: np.ndarray, r: float) -> np.ndarray:
    #ret = 1/similarity
    ret = 1/np.power(r, similarity)
    np.fill_diagonal(ret, 0)
    return ret
```

Phylogenetic Tree

Now we define some necessary functions to build the phylogenetic tree representations of these matrices.

- Function that creates trees using hierarchal clustering
  Input: numpy distance matrix, faculty names
  Output: plots a phylogenetic tree

```
def plot_dendrogram(distance_matrix, labels, linkage="single",
                    truncate_level=-1, figure_size=(20, 15),
                    dpi=300, savefig=False, fig_name="output.png"):
    # Set the parameters for the agglomerative clustering.
    cl = AgglomerativeClustering(distance_threshold=0, n_clusters=None, metric='precomputed',
                                 compute_full_tree=True, linkage=linkage, compute_distances=True)
    # Adjust the distance matrix to fit the clustering algorithm.
    cl.fit(distance_matrix)
    # Set the dendrogram plotting parameters
    kwarg = {"truncate_mode": "level", "p": truncate_level, "orientation": "right", "labels": labels}
        # for each merge initialization count in hierarchical clustering
    n_samples = len(cl.labels_)
    counts = np.zeros(cl.children_.shape[0])
    # Create a figure with the set size and dpi
    plt.figure(figsize=figure_size, dpi=dpi)
    #Determine the number of samples in each cluster by computing the counts for each merge.
    for i, merge in enumerate(cl.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count
# Create the linkage matrix using the counts, distances, and children.
        linkage_matrix = np.column_stack(
            [cl.children_, cl.distances_, counts]
        ).astype(float)
        # Plot the dendrogram using the linkage matrix and the specified parameters
        dendrogram(linkage_matrix, **kwarg)
        # Convert hierarchical clustering result to tree structure
        #tree = to_tree(linkage_matrix)
        pass
```

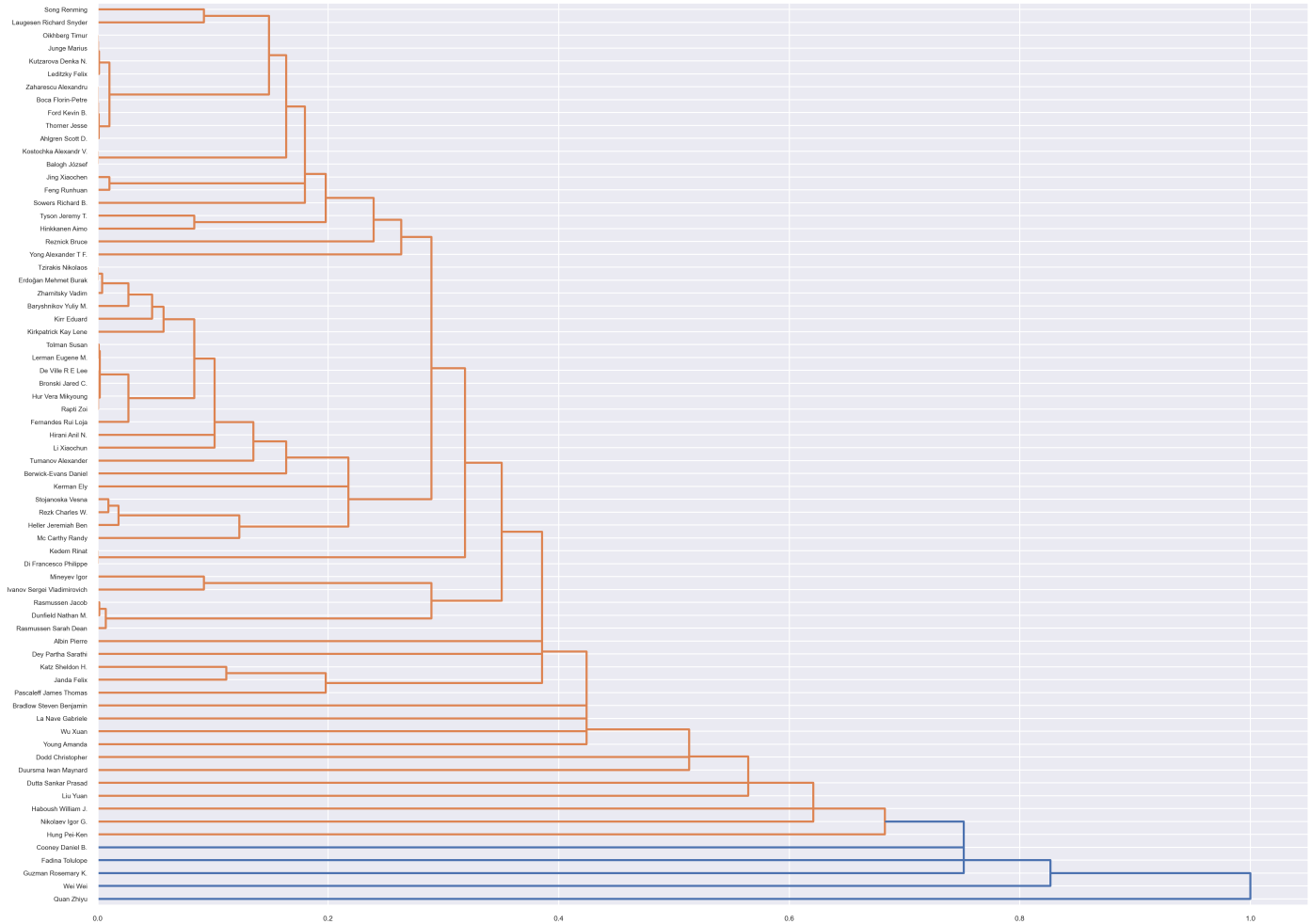
Phylogenetic Tree

We can now create the phylogenetic trees using our references data.

```
sim_com_ref = df[faculties].to_numpy() # Convert the similarity of common references into a numpy array
distance_matrix = d1(sim_com_ref,1.1) # Obtain the distance matrix

#print(distance_matrix)
plot_dendrogram(d1(sim_com_ref,1.1), faculties_no_underscores, linkage="single") #Plot the tree
```

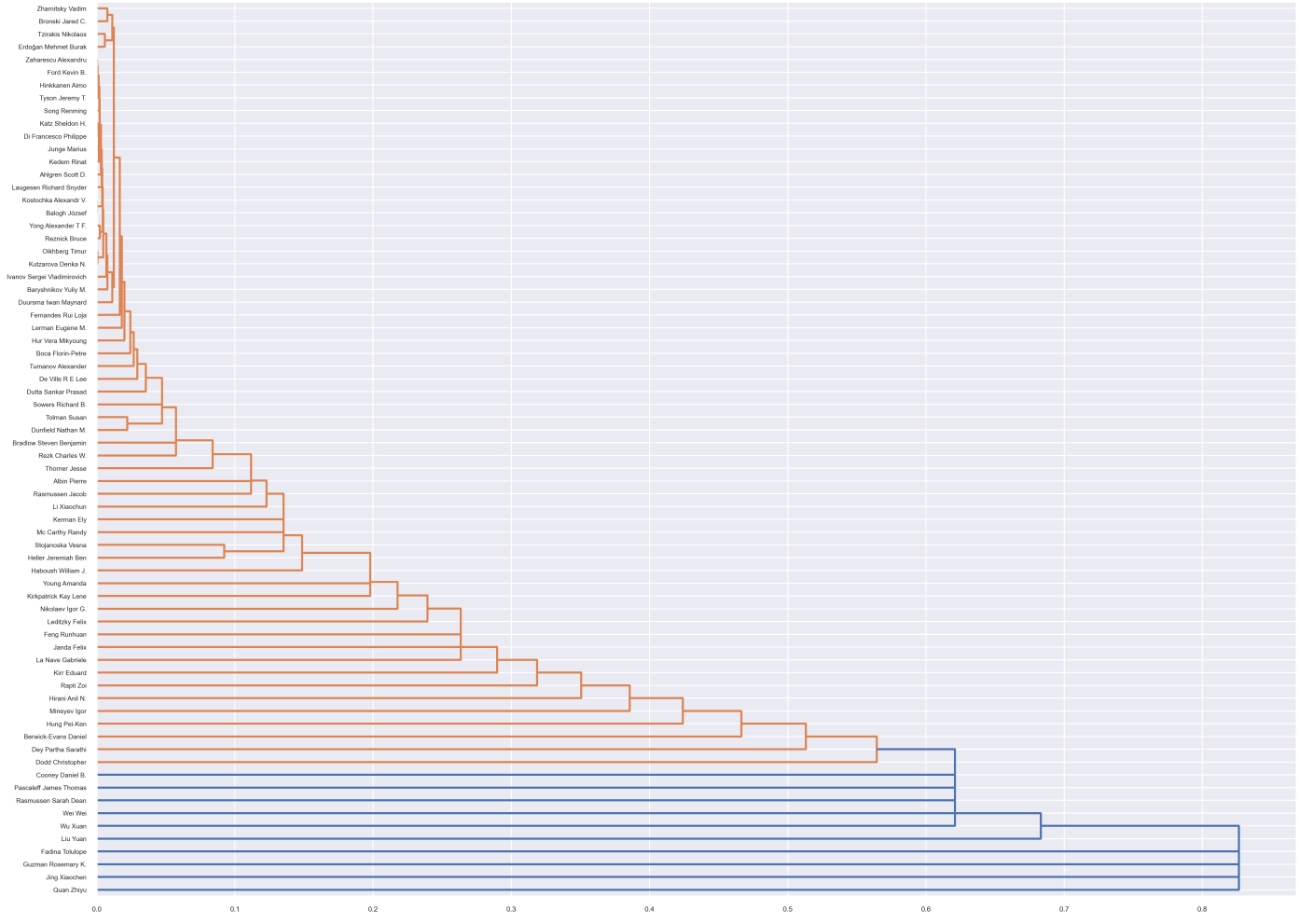
Phylogenetic Tree

We can now create the phylogenetic trees using our references data.



Phylogenetic Tree

We can now create the phylogenetic trees using our journals data.



Newick Format Tree

We will later need to write the tree in newick format to find the barycenters so we create a function for it now.

- Recursive function that writes a tree into newick format
  Inuput: The node we are recursing on, faculty names, height
  Output: A newick format tree

```
def tree_to_newick(node, labels, height):
    if node.is_leaf():
        return labels[node.id]
    # Recursive call for the node's left child
    left_str = tree_to_newick(node.left, labels, node.dist)
    # Recursive call for the node's right child
    right_str = tree_to_newick(node.right, labels, node.dist)
    # Construct the Newick format representation for the current node,(left_child:distance_to_left_child,right_child:distance_to_right_child)

    return f"({left_str}:{height - node.dist},{right_str}:{node.dist})"
```

Newick Format Tree

- Save hierarchical clustering result as Newick format tree
- Convert hierarchical clustering result to tree structure

```
# Save hierarchical clustering result as Newick format tree
from scipy.cluster.hierarchy import to_tree

linkage_matrix = linkage(distance_matrix, method='single', metric='euclidean')
# Convert hierarchical clustering result to tree structure
tree = to_tree(linkage_matrix)
```

Visualization

We also create heatmaps to visualize the distance matrices.

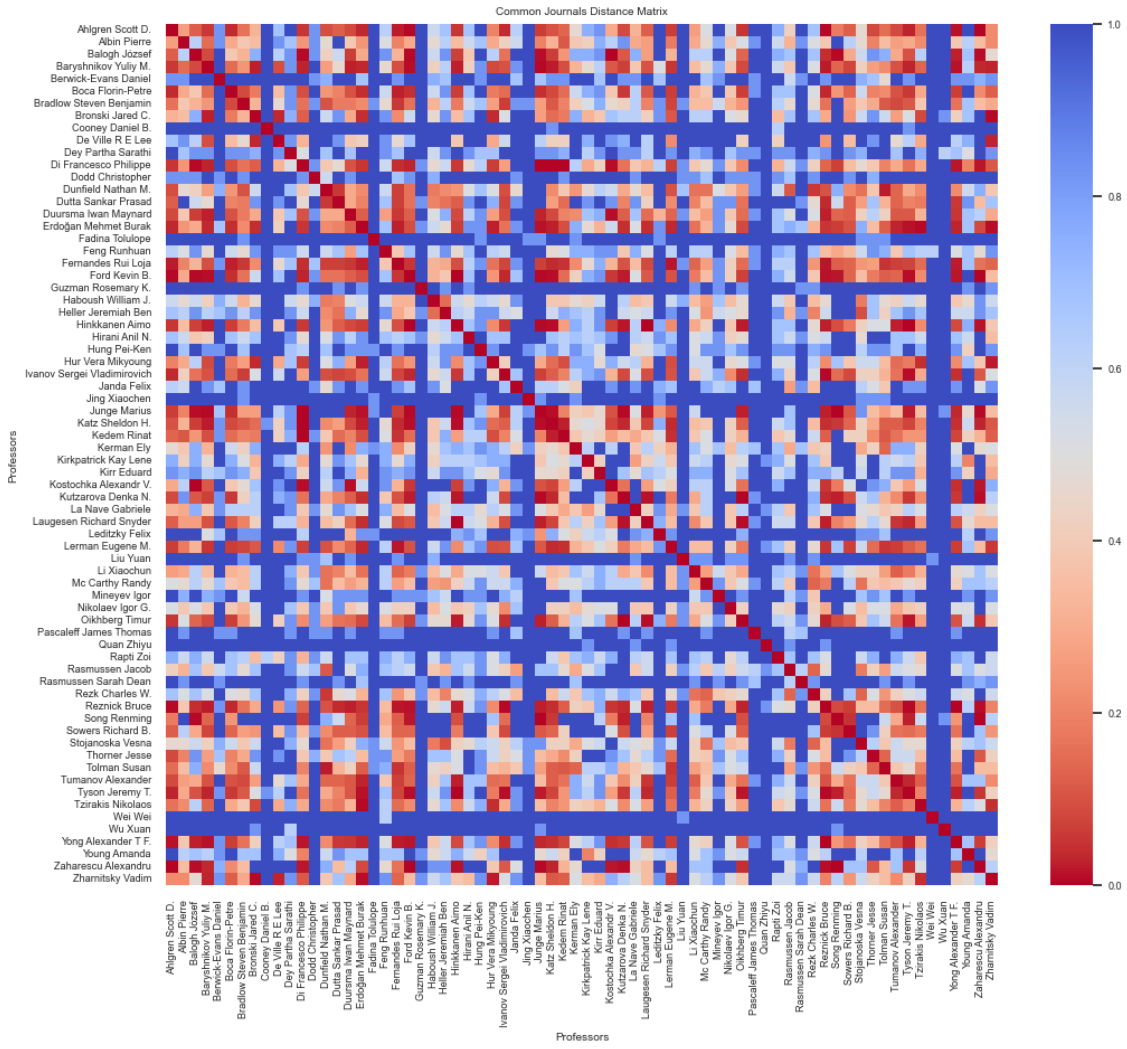
```
import matplotlib.pyplot as plt
import seaborn as sns
# Set the plot size and color mapping
plt.figure(figsize=(12, 10))
sns.heatmap(distance_matrix, cmap="coolwarm_r")
#Make the font smaller to fit nicely
sns.set(font_scale=0.6)

# We set the plot ticks as our list of faculty names
plt.xticks(ticks=np.arange(len(faculties_no_underscores)) + .5, labels=faculties_no_underscores, rotation=90)
plt.yticks(ticks=np.arange(len(faculties_no_underscores)) + .5, labels=faculties_no_underscores, rotation=360)

# General titles for clarity
plt.xlabel('Professors')
plt.ylabel('Professors')

plt.title('Common References Distance Matrix')

plt.show()
```

Visualization

For references distance matrices.

Visualization

For journals distance matrices.



Future Goal

1. Evaluate the resulting trees and implied clusterings for consistency with the existing department structure.
2. Aggregate the trees generated in the previous steps to plausible barycenters.
3. Use these barycenters to identify research clusters of the department.

Thank you

Thank you!