CS 6170 - Computational Topology

Tark Patel

May 7, 2021

1 Visualize real data

1.1 merge tree grapher

```
import math
import matplotlib.pyplot as plt
import networkx as nx
from networkx.drawing.nx_pydot import graphviz_layout
from matplotlib.backends.backend_agg import FigureCanvasAgg as
    FigureCanvas
from PIL import Image

def makeTickArray(array, threshold):
  tick_array = []
  last = float('inf')
  for v in array:
    if (abs(v-last) > threshold):
      tick_array.append(v)
    last = v
  return tick_array

def makeGraph(nodes, edges, node_values, x_values):
  G = nx.Graph()
  for node in nodes:
    G.add_node(node, pos=(x_values[node],node_values[node]))
  for edge in edges:
    G.add_edge(edge[0], edge[1])
  return G

def graphCompGraph(G, title="Complete Graph", labels=None):
  fig, ax = plt.subplots()
  canvas = FigureCanvas(fig)
  plt.title(title)
```

```
29    node_labels=nx.get_node_attributes(G,'label')
30    if node_labels:
31      nx.draw(G, with_labels=False, ax=ax)
32      nx.draw_networkx_labels(G,node_labels,font_size=16)
33    elif labels:
34      nx.draw(G, with_labels=False, ax=ax)
35      nx.draw_networkx_labels(G,labels,font_size=16)
36    else:
37      nx.draw(G, with_labels=True, ax=ax)
38    canvas.draw()
39    return Image.frombytes('RGB', fig.canvas.get_width_height(),canvas
          .tostring_rgb())
40
41
42  def graphMergeTree(G, title="Merge Tree", labels=None):
43    fig, ax = plt.subplots()
44    canvas = FigureCanvas(fig)
45    plt.title(title)
46
47    pos=nx.get_node_attributes(G,'pos')
48    node_labels=nx.get_node_attributes(G,'label')
49    node_values = [p[1] for p in pos.values()]
50    if node_labels:
51      nx.draw(G, pos, with_labels=False, ax=ax)
52      nx.draw_networkx_labels(G,pos,node_labels,font_size=16)
53    elif labels:
54      nx.draw(G, pos, with_labels=False, ax=ax)
55      nx.draw_networkx_labels(G,pos,labels,font_size=16)
56    else:
57      nx.draw(G, pos, with_labels=True, ax=ax)
58
59    ax.set_axis_on()
60    threshold = 0.01*(max(node_values)-min(node_values))
61    tick_array = makeTickArray(node_values, threshold)
62    ax.tick_params(left=True, labelleft=True)
63    plt.yticks(tick_array)
64    for val in tick_array:
65      plt.axhline(y=val, color='k', linestyle='-', linewidth=0.3)
66    canvas.draw()
67    return Image.frombytes('RGB', fig.canvas.get_width_height(),canvas
          .tostring_rgb())
68    # plt.show()
69
70  def getConnectivityValue(G, n1, n2):
71    val = float('-inf')
```

```
  for n in nx.shortest_path(G,n1,n2):
    val = max(val,G.nodes[n]['pos'][1])
  return val
```

1.2 obj loader

```
from pathlib import Path

def getConnectivityFromObj(path):
  vertices = []
  edges = []
  for line in open(path).readlines():
    items = line.rstrip('\n').split(' ')
    if items[0] == 'v':
      vertices.append([float(items[1]), float(items[2]), float(items
          [3])])
    elif items[0] == 'l':
      edges.append([int(items[1])-1, int(items[2])-1])
  return vertices, edges
```

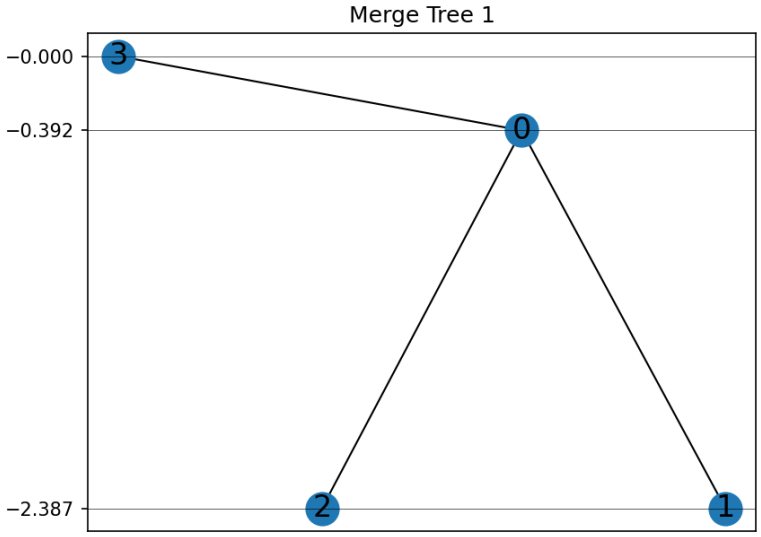
1.3 Load objs of 2 merge trees

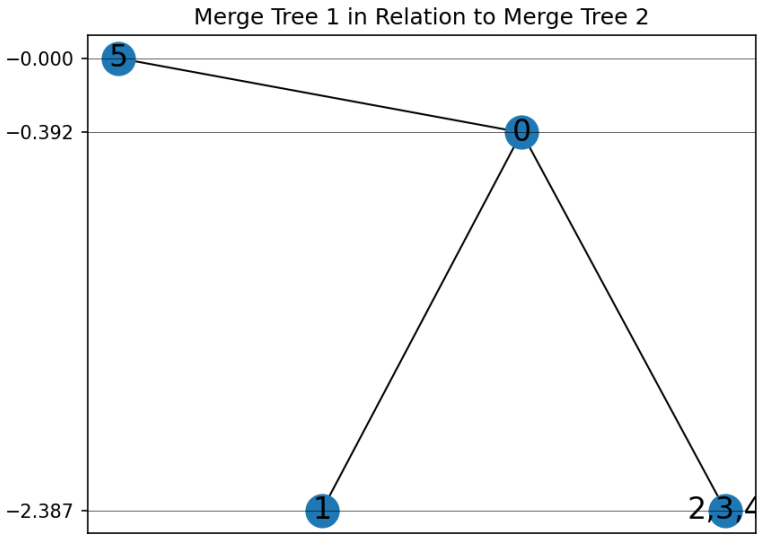
```
data_path = Path.home().joinpath('data')
m1_path = str(data_path.joinpath('MergeTree1.obj'))
m2_path = str(data_path.joinpath('MergeTree2.obj'))
[m1_vertices, m1_edges] = getConnectivityFromObj(m1_path)
[m2_vertices, m2_edges] = getConnectivityFromObj(m2_path)
m1_vertices = [[v[0],v[1],-v[2]] for v in m1_vertices]
m2_vertices = [[v[0],v[1],-v[2]] for v in m2_vertices]
```

1.4 Graph merge tree 1

```
import numpy as np

m1_nodes = set([v for edge in m1_edges for v in edge])
m1_node_values = np.array([])
m1_x_values = np.array([])
m1_og_labels = {0:"0", 1:"1", 2:"2", 3:"3"}
m1_as_m2_labels = {0:"0", 1:"2,3,4", 2:"1", 3:"5"}

for node in m1_nodes:
```

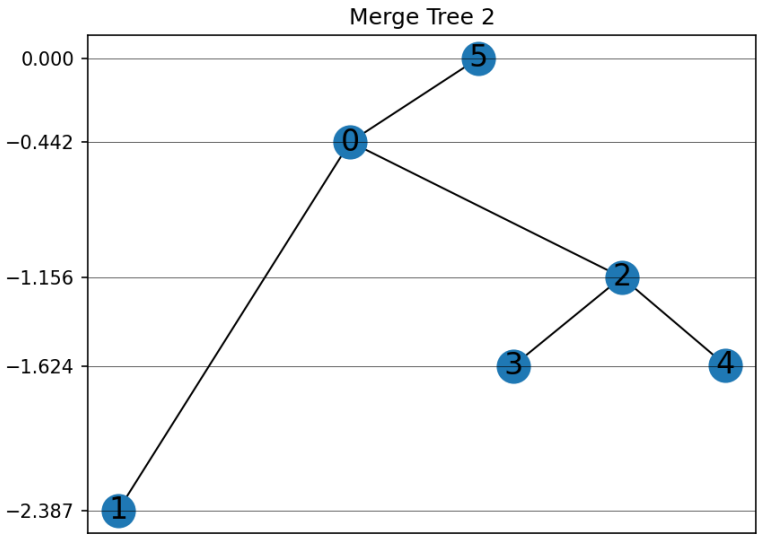
```
    val = m1_vertices[node]
    m1_node_values = np.append(m1_node_values, val[2])
    m1_x_values = np.append(m1_x_values, val[0])

G1 = makeGraph(m1_nodes, m1_edges, m1_node_values, m1_x_values)
graphMergeTree(G1, 'Merge Tree 1', m1_og_labels).save(data_path.
    joinpath("m1_og.png"))
graphMergeTree(G1, 'Merge Tree 1 in Relation to Merge Tree 2',
    m1_as_m2_labels).save(data_path.joinpath("m1_relation.png"))
```





1.5 Graph merge tree 2

```
m2_nodes = set([v for edge in m2_edges for v in edge])
m2_node_values = np.array([])
m2_x_values = np.array([])
m2_labels = {0:"0", 1:"1", 2:"2", 3:"3", 4:"4", 5:"5"}

for node in m2_nodes:
  val = m2_vertices[node]
  m2_node_values = np.append(m2_node_values, val[2])
  m2_x_values = np.append(m2_x_values, val[0])

G2 = makeGraph(m2_nodes, m2_edges, m2_node_values, m2_x_values)
graphMergeTree(G2, 'Merge Tree 2', m2_labels).save(data_path.
    joinpath("m2.png"))
```



1.6 Make x values for relative merge tree

```
m1_as_m2_x_values = np.empty(len(m2_x_values))
for v in m1_as_m2_labels.items():
  for node in v[1].split(","):
    m1_as_m2_x_values[int(node)] = m1_x_values[v[0]]
```

1.7 Convert merge tree to matrix

```
import numpy as np
np.set_printoptions(formatter={'float': lambda x: "{0:0.3f}".format(
    x)})

def buildConnectivityMatrix(G, labels, node_count):
  mat = np.zeros((node_count,node_count))
  node_lookup = np.zeros(node_count)

  for node in G.nodes:
    node_val = G.nodes[node]['pos'][1]
    for label_node in [int(n) for n in labels[node].split(',')]:
      mat[label_node][label_node] = node_val
      node_lookup[label_node] = node
```

```
13
14    for i in range(node_count):
15      for j in range(i):
16        mat[i,j] = mat[j,i] = getConnectivityValue(G,node_lookup[i],
              node_lookup[j])
17    return mat
18
19  induced_matrix1 = buildConnectivityMatrix(G1, m1_as_m2_labels, 6)
20  induced_matrix2 = buildConnectivityMatrix(G2, m2_labels, 6)
21
22  print("Induced Matrix 1 in Relation to Merge Tree 2:")
23  print(induced_matrix1)
24  print("Induced Matrix 2:")
25  print(induced_matrix2)
```

1.8 Convert induced matrix to graph

```
1   import math
2   import copy
3   import matplotlib.pyplot as plt
4   import numpy as np
5   from sortedcontainers import SortedSet
6   from igraph import Graph, EdgeSeq, plot
7   import networkx as nx
8   import pydot
9   from networkx.drawing.nx_pydot import graphviz_layout
10
11  def getGraphsOfMatrix(mat, x_values=None):
12      dim = mat.shape[0]
13
14      for i in range(1,dim):
15          for j in range(i):
16              mat[i,j] = mat[j,i]
17
18      node_values = mat.diagonal().tolist()
19
20      steps = SortedSet()
21      node_val_set = SortedSet()
22      for i in range(dim):
23          for j in range(i+1):
24              steps.add(mat[i,j])
25          node_val_set.add(mat[i,i])
26
27      values_without_nodes = steps - node_val_set
```

```
28
29      def get_val_of_edge(edge):
30          return mat[edge[0],edge[1]]
31
32      comp_graph = nx.Graph()
33      merge_tree = nx.Graph()
34      complete_graphs = []
35      merge_tree_graphs = []
36
37      unique_nodes = []
38      coalesced_nodes = []
39      edges_to_connect = []
40      values_without_nodes_by_step = []
41      # Evaluate each step of the connected components
42      for step in steps:
43
44          # Add data from step into lists
45          nodes_in_step = []
46          edges_in_step = []
47          coalesced_edges_in_step = []
48          coalesced_nodes_in_step_dict = {}
49          for i in range(dim):
50              for j in range(i+1):
51                  if (step == mat[i,j]):
52                      # The diagonal are all nodes
53                      if (i==j):
54                          nodes_in_step.append(i)
55                      # Above the diagonal are all connections
56                      else:
57                          # Coalesced nodes are ignored because they are
                                redundant
58                          if (i not in coalesced_nodes and j not in
                                coalesced_nodes):
59                              edges_in_step.append((i,j))
60                          else:
61                              coalesced_edges_in_step.append((i,j))
62
63          if (step in values_without_nodes):
64              values_without_nodes_by_step.append(step)
65
66          # Build complete graph
67          for node in nodes_in_step:
68              comp_graph.add_node(node)
69          for edge in edges_in_step:
70              comp_graph.add_edge(edge[0],edge[1])
```

```
71          for edge in coalesced_edges_in_step:
72              comp_graph.add_edge(edge[0],edge[1])
73          complete_graphs.append(copy.deepcopy(comp_graph))
74          # plt.title("Complete Graph of Step " + str(step))
75          # nx.draw(comp_graph, with_labels=True, arrows=False)
76          # plt.show()
77
78          for edge in edges_in_step:
79              # If nodes have an edge in the same step, they are
                    coalesced
80              if ((edge[0] in nodes_in_step) and (edge[1] in
                    nodes_in_step)):
81                  coalesced_nodes.append(edge[0])
82                  nodes_in_step.remove(edge[0])
83                  if edge[1] in coalesced_nodes_in_step_dict:
84                      coalesced_nodes_in_step_dict[edge[1]].append(edge
                            [0])
85                  else:
86                      coalesced_nodes_in_step_dict[edge[1]] = [edge[0]]
87                  # print("Nodes "+str(edge0)+" and "+str(edge1)+" are
                        coalesced")
88
89          # Remove any edges that contain a coalesced node
90          temp_edges_in_step = copy.deepcopy(edges_in_step)
91          for edge in temp_edges_in_step:
92              if (edge[0] in coalesced_nodes):
93                  edges_in_step.remove(edge)
94              elif (edge[1] in coalesced_nodes):
95                  edges_in_step.remove(edge)
96
97          # Any nodes that are not coalesced are added to the list of
                unique nodes
98          for node in nodes_in_step:
99              unique_nodes.append(node)
100
101         # Remove edges that are connected indirectly
102         temp_edges_in_step = copy.deepcopy(edges_in_step)
103         for edge1 in temp_edges_in_step:
104             for edge2 in temp_edges_in_step:
105                 if (edge1 != edge2):
106                     node_set = SortedSet([edge1[0], edge1[1], edge2
                            [0], edge2[1]])
107                     # If there are 3 unique nodes, than the edges
                            might form a triangle
108                     if (len(node_set) == 3):
```

```
109                     node_value_set = SortedSet([(node_values[
                            node_set[0]],node_set[0]),
110                                                 (node_values[node_set
                                                        [1]],node_set[1])
                                                        ,
111                                                 (node_values[node_set
                                                        [2]],node_set[2])
                                                        ])
112                     # If there are only 2 unique node values, then
                             min and mid nodes were not coalesced,
                            therefore cannot contain an edge
113                     if (len(node_value_set) == 3):
114                         largest = max(node_value_set)
115                         smallest = min(node_value_set)
116                         mid = (node_value_set - SortedSet([largest,
                                 smallest]))[0]
117
118                         # Remove whichever edge connects to the
                                node lower with the lowest value
119                         edge1_tallest = smallest[1] in edge1 and
                                mat[edge1] > largest[0]
120                         edge2_tallest = smallest[1] in edge2 and
                                mat[edge2] > largest[0]
121                         # Handles special case where the min and
                                mid are connected by a value without a
                                node
122                         if (edge1_tallest or edge2_tallest):
123                             if ((smallest[1], mid[1]) in
                                    edges_in_step):
124                                 edges_in_step.remove((smallest[1],
                                        mid[1]))
125                             elif ((mid[1], smallest[1]) in
                                    edges_in_step):
126                                 edges_in_step.remove((mid[1],
                                        smallest[1]))
127                         # Handles normal cases
128                         elif ((smallest[1], mid[1]) in
                                edges_to_connect or (mid[1], smallest
                                [1]) in edges_to_connect):
129                             if (smallest[1] in edge1 and edge1 in
                                    edges_in_step):
130                                 edges_in_step.remove(edge1)
131                             elif (smallest[1] in edge2 and edge2 in
                                     edges_in_step):
132                                 edges_in_step.remove(edge2)
```

```
133
134         # Any remaining edges meet the criteria, so they are added to
                the list of edges that should appear in the graph
135         for new_edge in edges_in_step:
136             edges_to_connect.append(new_edge)
137
138         # Values without nodes are considered for the edges that need
                to be connected
139         # The node index for the drawing list is kept track
140         # Values without nodes are denoted with None type
141         edges_to_draw = []
142         edges_to_draw_node_index = []
143         for edge in edges_in_step:
144             fro = node_values[edge[1]]
145             to = node_values[edge[0]]
146             connection = get_val_of_edge(edge)
147             if (to == connection or fro == connection):
148                 edges_to_draw.append((to, fro))
149                 edges_to_draw_node_index.append(edge)
150             elif (connection in values_without_nodes):
151                 edges_to_draw.append((to, connection))
152                 edges_to_draw.append((connection, fro))
153                 none_node_name = "None-"+str(connection)
154                 edges_to_draw_node_index.append((edge[0],
                        none_node_name))
155                 edges_to_draw_node_index.append((none_node_name, edge
                        [1]))
156
157         # Draw merge tree
158         for node in nodes_in_step:
159             label = str(node)
160             if node in coalesced_nodes_in_step_dict:
161                 for coalesced_node in coalesced_nodes_in_step_dict[
                        node]:
162                     label = label + "," + str(coalesced_node)
163             if type(x_values) == None:
164                 merge_tree.add_node(node, pos=(np.random.rand(),
                        node_values[node]),label=label)
165             else:
166                 merge_tree.add_node(node, pos=(x_values[node],
                        node_values[node]),label=label)
167         for value in values_without_nodes_by_step:
168             merge_tree.add_node("None-"+str(value), pos=(np.random.
                    rand(),value))
169         for edge in edges_to_draw_node_index:
```

```
            merge_tree.add_edge(edge[0], edge[1])
        merge_tree_graphs.append(copy.deepcopy(merge_tree))
    return complete_graphs, merge_tree_graphs, steps
```

1.9 Get data of both merge trees

```
[complete_graphs1, merge_tree_graphs1, steps1] = getGraphsOfMatrix(
    induced_matrix1, m1_as_m2_x_values)
[complete_graphs2, merge_tree_graphs2, steps2] = getGraphsOfMatrix(
    induced_matrix2, m2_x_values)
```

1.10 Make gif

```
intervals = 50
pause = 20
merge_trees = []
ims = []
for i in range(intervals):
    w = float(i/(intervals-1.0))
    interp_mat = (1.0-w)*induced_matrix1 + w*induced_matrix2
    interp_x_values = (1.0-w)*np.array(m1_as_m2_x_values) + w*np.
        array(m2_x_values)
    [interp_complete_graphs, interp_merge_tree_graphs, steps2] =
        getGraphsOfMatrix(interp_mat, interp_x_values)
    merge_trees.append(interp_merge_tree_graphs[-1])
    ims.append(graphMergeTree(interp_merge_tree_graphs[-1], 'Merge
        Tree Linear Interpolation'))
ims_rev = copy.deepcopy(ims)
ims_rev.reverse()
ims = ims+[ims_rev[0].copy() for i in range(pause)]+ims_rev+[ims[0].
    copy() for i in range(pause)]
dur = [10 for i in range(len(ims))]
dur[-1] = 100
ims[0].save(data_path.joinpath('mergeTree.gif'), duration=dur,
    save_all=True, append_images=ims[1:],loop=0,optimize=False)
```

This is a gif. You will need to open this outside of the document.

1.11 Write data to network file

```
for i in range(intervals):
```

```
2      nx.write_gpickle(merge_trees[i], data_path.joinpath("
           mergeTreeLinearInterp","step_"+str(i)+".gpickle"))
```