Practical 6: LCA Mapping

Ximena Moure Eliya Tiram

24/10/2023, submission deadline 30/10/2023

This code is used to build the restricted tree whcih will be used in the exercises of this lab.

```python
import networkx as nx


def load_graph_from_file(file_path):
    G = nx.DiGraph()  # Initialize directed graph
    with open(file_path, 'r') as file:
        for line in file:
            child, parent, rank = map(str.strip, line.split('|')[:3])
            child, parent = int(child), int(parent)

            # Skip self-loops
            if child != parent:
                # Add edge
                G.add_edge(parent, child)
                # Assign rank attribute to the child node
                G.nodes[child]['rank'] = rank
    return G


file_path = "nodes.dmp"

initial_tree = load_graph_from_file(file_path)

valid_ranks
    = ["superkingdom", "phylum", "class", "order", "family", "genus", "species"]
# Filter nodes based on rank attribute
nodes_to_remove = [node for node, attrs
    in initial_tree.nodes(data=True) if attrs.get('rank') not in valid_ranks]
# Create a copy of the graph
restricted_taxonomy = initial_tree.copy()


def create_restricted_tax(graph, nodes_to_delete):
    for node in nodes_to_delete:
        successors = list(graph.successors(node))
        predecessors = list(graph.predecessors(node))
        if node != 1:
            graph.remove_node(node)
        for source in predecessors:
            for target in successors:
                if source != target and not graph.has_edge(source, target):
                    graph.add_edge(source, target)


create_restricted_tax(restricted_taxonomy, nodes_to_remove)
print('Tree: ', nx.is_tree(restricted_taxonomy))
print('
   Rooted:', nx.is_arborescence(restricted_taxonomy))  # to check if it is rooted
print('Number
   of nodes in the restricted taxonomy: ', restricted_taxonomy.number_of_nodes())
print('Number
    of edges in the restricted version: ', restricted_taxonomy.number_of_edges())

```

Listing 1: Python code to extract restricted taxonomy

1. **Given a file nodes.dmp for the NCBI taxonomy and a file mapping.txt of mappings of sequence reads to NCBI taxonomic identifiers, write a Python script to find the LCA mapping for each sequence read.**

Give the code of your Python script as your answer to this question, using the LATEX package listings.

```
def find_lineage(taxonomy_graph, node, cache=None):
    if cache and node in cache:
        return cache[node]

    lineage = []
    current = node
    while current in taxonomy_graph:
        lineage.append(current)
        parents = list(taxonomy_graph.predecessors(current))
        if not parents:
            break
        current = parents[0]

    lineage.reverse()

    if cache is not None:
        cache[node] = lineage

    return lineage


# Reading the mapping file
mapping = {}
with open('mapping.txt', 'r') as file:
    for line in file:
        li = line.strip().split()
        sq_read_id = li[0]
        nodes = [int(node) for node in li[1:]]
        mapping[sq_read_id] = nodes


lineages = {}
lineage_cache = {}
for read_id, taxonomic_nodes in mapping.items():
    lineage_for_read = [find_lineage
   (restricted_taxonomy, node, lineage_cache) for node in taxonomic_nodes]
    lineages[read_id] = lineage_for_read


# LCAs for each read
LCAs = {}

for read_id, seq in lineages.items():
    min_length = min(len(lst) for lst in seq)
    lst1 = seq[0]

    i = 0
    while i < min_length:
        current_item = lst1[i]
        # Check if the current item is the same for all sequences
        all_same = all(lst[i] == current_item for lst in seq[1:])

        # If any sequence differs at this position, break
        if not all_same:
            break

        i += 1

    # Store the last common ancestor (i.e., the item before the divergence)
    LCAs[read_id] = lst1[i - 1] if i > 0 else None
```

```
print("The LCAs for each read: \n", LCAs)

# Retrieve the rank for each LCA
lca_ranks = {read_id: restricted_taxonomy.
    nodes[lca].get('rank') for read_id, lca in LCAs.items() if lca is not None}


highest_rank = None
lowest_rank = None

# Determine the highest rank
for rank in lca_ranks.values():
    if highest_rank
    is None or valid_ranks.index(rank) < valid_ranks.index(highest_rank):
        highest_rank = rank

print("Highest taxonomic rank for the LCAs:", highest_rank)

# Retrieve the rank for each LCA if it's not None
lca_ranks = {read_id: restricted_taxonomy.
    nodes[lca].get('rank') for read_id, lca in LCAs.items() if lca is not None}

# Determine the lowest rank
for rank in lca_ranks.values():
    if lowest_rank
    is None or valid_ranks.index(rank) > valid_ranks.index(lowest_rank):
        lowest_rank = rank

print("Lowest taxonomic rank for the LCAs:", lowest_rank)

```

Listing 2: Python code ex1

Result is shown in the image below (Figure: 1).

```
The LCAs for each read:
 {'R00010': 1, 'R00020': 1, 'R00030': 1, 'R00040': 1, 'R00050': 1, 'R00060': 1, 'R00070': 1, 'R00080': 1, 'R00090': 1, 'R
```

Figure 1: Ex1 result

2. **What is the highest taxonomic rank (that is, toward kingdom) for these LCA mappings?**
   It is no rank.

3. **What is the lowest taxonomic rank (that is, toward species) for these LCA mappings?**

   It is no rank

4. **Given a file nodes.dmp for the NCBI taxonomy and a file mapping.txt of mappings of sequence reads to NCBI taxonomic identifiers, write a Python script to find the optimal (in terms of the F-measure) taxonomic assignment for each sequence read. Give the code of your Python script as your answer to this question, using the LATEX package listings**

   In order to do this part, we modified the code to get the skeleton tree that we delivered for the previous lab. The new code is the following

```
def build_skeleton_tree(lineages):
    skeleton_graph = nx.DiGraph()

    for lineage in lineages:
        # Add nodes and edges from the lineage to the skeleton graph
```

```
        skeleton_graph.add_nodes_from(lineage)
        skeleton_graph.add_edges_from(zip(lineage[:-1], lineage[1:]))

    return skeleton_graph


skeleton_trees = {}

for read_id, seq_lineages in lineages.items():
    skeleton_trees[read_id] = build_skeleton_tree(seq_lineages)


def compressLinearPaths(tree):
    """
    Contract elementary paths in the tree.
    Elementary
    paths are paths where a node has only one child and can be contracted.
    This function traverses the tree in a bottom-up manner.
    """
    contracted_tree = tree.copy()
    nodes_to_process = [node for node in nx.
   topological_sort(contracted_tree) if contracted_tree.out_degree(node) == 1]

    for current_node in nodes_to_process:
        parent = next(contracted_tree.predecessors(current_node), None)
        child = next(contracted_tree.successors(current_node), None)

        # Skip if it's a root or leaf node
        if parent is None or child is None:
            continue

        # Contract the node by connecting its parent to its child
        contracted_tree.add_edge(parent, child)
        contracted_tree.remove_node(current_node)

    return contracted_tree


contracted_skel_trees = {}
# Counting nodes after contracting skeleton trees
nodes_per_read = {}
for read_id, lca_tree in skeleton_trees.items():
    contracted_skel_tree = compressLinearPaths(lca_tree)
    contracted_skel_trees[read_id] = contracted_skel_tree
    num_nodes = len(contracted_skel_tree.nodes)
    nodes_per_read[read_id] = num_nodes


print("Details of each skeleton tree:")
for read_id, tree in contracted_skel_trees.items():
    num_nodes = tree.number_of_nodes()
    num_edges = tree.number_of_edges()
    print(f"Read
   ID: {read_id}, Number of Nodes: {num_nodes}, Number of Edges: {num_edges}")


```

Listing 3: Python code for building the LCA skeleton tree for each sequence read

```
Details of each skeleton tree:
Read ID: R00010, Number of Nodes: 16, Number of Edges: 15
Read ID: R00020, Number of Nodes: 32, Number of Edges: 31
Read ID: R00030, Number of Nodes: 44, Number of Edges: 43
Read ID: R00040, Number of Nodes: 58, Number of Edges: 57
Read ID: R00050, Number of Nodes: 70, Number of Edges: 69
Read ID: R00060, Number of Nodes: 82, Number of Edges: 81
Read ID: R00070, Number of Nodes: 99, Number of Edges: 98
Read ID: R00080, Number of Nodes: 106, Number of Edges: 105
Read ID: R00090, Number of Nodes: 129, Number of Edges: 128
Read ID: R00100, Number of Nodes: 135, Number of Edges: 134
```

Figure 2: Skeleton tree for each sequence read

Now, we can do exercise 4. The code is the following

```
1
2  def calculate_precision_recall(tp, fp, fn):
3      precision = tp / (tp + fp) if (tp + fp) > 0 else 0
4      recall = tp / (tp + fn) if (tp + fn) > 0 else 0
5      return precision, recall
6
7
8  def calculate_f_measure(precision, recall):
9      return 2/(1/precision + 1/recall)
10
11
12 # Optimal calculation for each sequence read
13 results = []
14 optimal_ranks = {}
15
16 for read_id in mapping.keys():
17     # Retrieve the contracted skeleton tree for the read
18     tree = contracted_skel_trees.get(read_id, None)
19
20     # Check if the tree exists
21     if tree is None:
22         print(f"No skeleton tree found for read {read_id}")
23         continue
24
25     max_node = ''
26     max_F = 0
27     max_rank = ''
28     for node in tree.nodes():
29         # Calculate F_measure for each internal node
30         if node not in mapping[read_id]:
31             descendants_of_node = set(nx.descendants(tree, node))
32
       true_positives = len(descendants_of_node.intersection(mapping[read_id]))
33             false_negatives = len(mapping[read_id]) - true_positives
34             descendants = 0
35             for d in set(nx.descendants(restricted_taxonomy, node)):
36                 if restricted_taxonomy.out_degree(d) == 0:
37                     descendants += 1
38             false_positives = descendants - true_positives
39             precision, recall = calculate_precision_recall
   (true_positives, false_positives, false_negatives)
40             f_measure = calculate_f_measure(precision, recall)
41
42             # Get the max F_measure
```

```
            if f_measure > max_F:
                max_F = f_measure
                max_node = node
                max_rank = restricted_taxonomy.nodes[node].get('rank')

    r = f"The optimal assignment of {read_id} is {max_node} with F_measure = {max_F}, taxonomic rank {max_rank} "
    results.append(r)
    optimal_ranks[read_id] = max_rank  # Store the rank

# Print results
print("OPTIMAL TAXONOMIC ASSIGNMENTS")
for result in results:
    print(result)


highest_rank = min(optimal_ranks.values(), key=lambda x: valid_ranks.index(x))
print(f"Highest taxonomic rank among optimal assignments: {highest_rank}")

lowest_rank = max(optimal_ranks.values(), key=lambda x: valid_ranks.index(x))
print(f"Lowest taxonomic rank among optimal assignments: {lowest_rank}")
```

Listing 4: Python code for ex 4

Results after running the code are shown in the image below (Figure 3)

```
OPTIMAL TAXONOMIC ASSIGNMENTS
The optimal assignment of R00010 is 7088 with F_measure = 3.289419581914771e-05, taxonomic rank order
The optimal assignment of R00020 is 7100 with F_measure = 0.0005665722379603399, taxonomic rank family
The optimal assignment of R00030 is 7458 with F_measure = 0.0012606366214938543, taxonomic rank family
The optimal assignment of R00040 is 953 with F_measure = 0.0011940298507462687, taxonomic rank genus
The optimal assignment of R00050 is 10193 with F_measure = 0.005641748942172073, taxonomic rank family
The optimal assignment of R00060 is 7100 with F_measure = 0.0005633802816901409, taxonomic rank family
The optimal assignment of R00070 is 384774 with F_measure = 0.01990049751243781, taxonomic rank genus
The optimal assignment of R00080 is 4747 with F_measure = 0.0003652634462606155, taxonomic rank family
The optimal assignment of R00090 is 337673 with F_measure = 0.017699115044247787, taxonomic rank family
The optimal assignment of R00100 is 9397 with F_measure = 0.0018009905447996398, taxonomic rank order
Highest taxonomic rank among optimal assignments: order
Lowest taxonomic rank among optimal assignments: genus
```

Figure 3: Results ex 4

5. **What is the highest taxonomic rank (that is, toward kingdom) for these taxonomic assignments?**
   The highest is order.

6. **What is the lowest taxonomic rank (that is, toward species) for these taxonomic assignments?**
   The lowest is genus.