1 Course Project

Taxonomic identification and phylogenetic profiling software TIPP [1] requires solving a phylogenetic placement problem in order to insert a new alignment into a tree. Currently, software such as pplacer [2] is an accurate, maximum likelihood phylogenetic placement method. However, pplacer does not scale to problems larger than 1,000 sequences. Therefore, alternative phylogenetic placement methods, such as APPLES [3], were developed for solving phylogenetic placement. While APPLES has been tested to run on 200,000 sequences, the accuracy of the method for smaller problems is typically worse than pplacer. This course project aims to improve phylogenetic placement software already available in two different approaches. The first proposed approach runs pplacer in a divide-and-conquer scheme to determine the optimal placement of the query sequence. The second approach, however, uses APPLES to identity the most probable region for a query sequence. Once this region is identified, pplacer is used on some collection of clades around this region of the tree in order to determine a more accurate placement.

1.1 Approach 1

Approach 1 aims to solve the placement problem by using a divide-and-conquer approach, using pplacer to solve the smaller phylogenetic placement problems. The first step of this approach is to divide the backbone tree into problems small enough that pplacer can readily be used. The software will divide the backbone into disjoint subtrees with fewer than 1000 sequences each. This will be done similarly to the first step of PASTA described in [4], using centroid decomposition. A centroid decomposition approach is used to determine subtrees smaller than some prescribed size, such as 1000 sequences. The centroid decomposition determines a centroid edge in the tree that partitions the tree into two halves of roughly equal size. Each subtree is then recursively subdivided using this same approach, until each subtree has no more sequences that the prescribed limit. We will start with the maximum size of the subtrees at 1000 sequences. We will also consider if it may improve the accuracy to have smaller trees, such as 500 sequences, though based on the placement accuracy by reference tree size in the comparison to APPLES in [3], we expect that accuracy will be higher for larger reference trees. After decomposing the tree via centroid decomposition, the query sequence will be placed into each of the subtrees generated. Pplacer will run independently on each of the subtrees from this centroid decomposition. Since each subproblem is independent of the other subproblems, this step in the algorithm is embarassingly parallel.

After determining the placement on each subtree, the placement of the query sequence on the subtree is used as the placement of the query sequence into the backbone tree. Each backbone tree returned from this method can then be compared, and the one with the best score is taken as the output from the algorithm. Each tree is scored using RAxML in fixed-tree mode, allowing us to compare the maximum likelihood score of each of the placement trees. Based on the location of the best score on the best subtree, the algorithm returns the corresponding location on the original backbone tree. Pseudocode for approach 1 is shown below in algorithm 1.

1.2 Approach 2

Approach 2 aims to combine the scalability of APPLES with the accuracy empirically seen for pplacer on smaller subtrees. Initially, APPLES is run on the query sequence and the entire backbone tree. The placement returned from APPLES is then used to identify a collection of (relatively) small clades containing this placement. Each clade is chosen to be smaller than some prescribed size, such as 1000 sequences. Since there is not a single unique clade satisfying this property, some (random) collection of clades is considered. If time permits, a more sophisticated method for determining clades in the backbone tree containing the query sequence may be considered. Psuedocode for generating a single (random) clade are shown below in algorithm 3.6. Balaban et al. observed that, for the RNASim-VS data set, the delta errors in APPLES's placement are likely to be relatively small. This indicates that the placement of the query sequence into the tree may be quite close to the true placement, meaning that only a small refinment is needed to produce the correct result. Pplacer is used on each of the clades identified in the previous step. Note that, similar to the previous approach, the placement and scoring on each clade is completely independent of the other clades. This makes this portion of the algorithm embarrasingly parallel. As in the other approaches, the final result is the placement of the query sequence in the original backbone tree based on its subtree placement. The placement that minimizes the maximum likelihood scores, as computed by RAxML in fixed tree mode, is then chosen as the placement returned by the algorithm. Note also that the original placement, as determine by APPLES, is also included in this consideration. Pseudocode for approach 2 is shown below in algorithm 2.

2 Data

2.1 Testing Data

We will use the 1000M1 dataset from [5]. This includes 20 replicate datasets.

2.2 Benchmarking Data

For simulated data, we will use the same datasets as APPLES used from [6], and use the RNASim-VS sets. These sets are of size 500, 1000, 5000, 10,000, 50,000, 100,000 and 200,000. For each of the sizes, there are a total of 5 replicates. The 200,000 sequence data set, however, only has a single replicate.

3 Assessment

3.1 Asessment Criteria

We will be comparing to both pplacer and APPLES for accuracy and runtime. For small backbone trees (less than 1000 sequences), our approach will equivalent to pplacer, so the accuracy at that point should be equal with no meaningful increase in runtime. For larger trees, we will look at the error rate relative to APPLES and at how the implementation scales. To calculate the error, we will use the same delta error calculation as APPLES to compare the sets of bipartions for the tree with the placement from our system and the bipartion from the tree with the correct placement. In concrete terms, delta error is defined as

$$\Delta e(P) = |B(T^*)\backslash B(P)| - |B(T^* \restriction_{\mathcal{L}})\backslash B(T)|,$$

where $\mathcal{L}$ denotes the leafset, P denotes the tree after placement, T^* denotes the true tree on $\mathcal{L} \cup \{q\}$, $T^* \restriction_{\mathcal{L}}$ denotes the true tree restricted to $\mathcal{L}$, and $B(\cdot)$ denotes the bipartition set of a tree [3].

Similar to Balaban and coworkers [3], we plan to consider a leave-one-out strategy in testing and evaluating the 1000M1 and RNASim-VS datasets. The leave-one-out strategy starts with the true tree, T, and identifies a random leaf node q from the tree. This leaf node is removed and then added back into the tree T using the placement software. This process is repeated 200 times to generate a distribution of errors produced by the placement software.

3.2 Analysis Hardware

We will perform the analysis on NCSA's Blue Waters. Each Blue Waters XE node has 16 cores. As neither APPLES nor pplacer has GPU capabilities, we will use the XE nodes instead of the XK nodes. Development and small testing will be conducted on the campuscluster or monza, a 12 core workstation.

3.3 Software

For comparison and steps of the implementation, we will be using both APPLES and pplacer. For the implementation, we will also use RAxML [7].

We will use APPLES version 1.3.0, which requires Python 3. The code can be found at:
https://github.com/balabanmetin/apples. We will only be running APPLES with alignment, as pplacer requires an alignment.

The command for APPLES will be:

```
python3 ~/apples/run_apples.py -t backbone.nwk -s aln_dna.fa -q query.fa -T 16 -o app
```

We will use pplacer version v1.1.alpha19-0-g807f6f3. The binary files can be found at:
https://github.com/matsen/pplacer/releases/tag/v1.1.alpha19.

The command for pplacer will be:

```
pplacer -m GTR -s RAxML_info.REF -t backbone.nwk -o query.jplace aln_dna.fa -j 1
```

We will use the sequential RAxML version 8.2.12. Parallelism with RAxML will be done through calling RAxML in a multithreaded section. This will be used to compare different pplacer outputs in both approaches. The maximum likelihood score returned from RAxML in fixed tree mode is used as the score for the two approaches. The command for RAxML-NG will be:

```
./raxml-ng --evaluate --model GTR+G --tree test.tree
```

3.4 Time Estimate

Given that pplacer costs 0.25 CPU minutes and APPLES costs 0.04 CPU minutes for a tree of size 1000 [3], then we estimate the following CPU times when pplacer is run on 5 subtrees. For each tree size there are 5 replicate datasets, except for the tree size of 200,000, which has 1 replicate, and the 1000M1 dataset, which has 20 replicates.

We multiply our estimate by 1.5 in order to account for other overhead cost not included within the time for pplacer or APPLES. As we are using RAxML in fixed tree mode for both approaches, it should not add signficiant time, and increased estimate should include the time for RAxML. In addition, there is a small overhead to the centroid decomposition technique used by approach 1 and the clade identification used by approach 2. Estimated compute resources for both approaches are shown below in tables 1, 2.

Approach 1	Data set	CPU time/replicate (min)	CPU Time
	500	0.94	4.69
	1000	1.88	9.38
	5000	9.38	46.88
	10000	18.75	93.75
	50000	93.75	468.75
	100000	187.5	937.5
(1 replicate)	200000	375	375
1000M1 (20 replicates)	1000	1.88	37.5
		CPU Time (hr):	32.27

Table 1: Estimated computational resources needed for approach 1. Note that a 1.5x safety factor is built in to the time estimate to account for RAxML scoring on fixed trees and centroid decomposition.

Together, the two approaches result in a total of 65.6 CPU hours. As each node has 16 CPUs, allowing APPLES to run on 16 cores and pplacer to run with the different subtrees simultaneously, the actual time required may be as low as 5 node hours.

Approach 2	Data set	CPU time/replicate (min)	CPU Time (min)
RNASim-VS	500	0.97	4.84
	1000	1.94	9.68
	5000	9.68	48.38
	10000	19.35	96.75
	50000	96.75	483.75
	100000	193.5	967.5
(1 replicate)	200000	387	387
1000M1 (20 replicates)	1000	1.94	38.7
		CPU Time (hr):	33.30

Table 2: Estimated computational resources needed for approach 2. Note that a 1.5x safety factor is built in to the time estimate to account for RAxML scoring on fixed trees and clade identification. A fixed number of 5 clades are considered for the purposes of estimating compute resources.

3.5 Approach 1 Pseudocode

Algorithm 1: divide-and-conquer pplacer

```
Result: $T'$, tree T with query sequence $q$ added
Input: Tree $T$ on $N$ sequences, the MSA of $N + 1$ sequences, and
       query sequence $q$
// centroidDecomposition decomposes a tree into roughly equal size,
  disjoint parts until the trees are no larger than the prescribed size.;
// modifyTree adds sequence to a tree based on the sequence's
  location in the a subtree with the sequence added;
$\{T_1, \dots, T_n\} \leftarrow$ centroidDecomposition($T$, 1000);
$\{S_1, \dots, S_n\} \leftarrow 0$ // Score for each tree;
parallel for $i = 1, \dots, n$ do
    // Place query sequence $q$ into the subtree;
    $T'_i \leftarrow$ pplacer($T_i, q$);
    // Add the location of the query sequence $q$ to a copy of $T$;
    $T_{q_i} \leftarrow$ modifyTree($T, T'_i, q$);
    // RAxMLScorer runs RAxML in fixed tree mode.;
    // The output score is the maximum likelihood found on the
      tree.;
    $S_i \leftarrow$ RAxMLScorer($T_{q_i}$);
endfor
// Do a maxLoc reduction for the tree;
$bestTreeIndex \leftarrow \text{argmax}_i(S_1, \dots, S_n)$;
return $T'_{q_{bestTreeIndex}}$;
```

3.6 Approach 2 Pseudocode

Algorithm 2: APPLES with pplacer

```
Result: $T'$, tree T with query sequence $q$ added
Input: Tree $T$ on $N$ sequences, the MSA of $N+1$ sequences, query
       sequence $q$, and the number of clades considered, $N_{clades}$.
// modifyTree adds sequence to a tree based on the sequence's
  location in the a subtree with the sequence added;
// getRegion finds a subtree containing the sequence with a
  maximum number of sequences;
// Run APPLES;
$T'_{APPLES} \leftarrow$ APPLES($T, q$);
// It could be the case that APPLES produces the best score;
$S_{apples} \leftarrow$ RAxMLScorer($T'_{apples}$);
$\{S_1, \ldots, S_{N_{clades}}\} \leftarrow 0$ // Score for each clade;
parallel for $i = 1, \ldots, N_{clades}$ do
    // Identify (random) region of T where q was placed with fewer
      than 1000 sequences;
    $T_i \leftarrow$ getRegion($T'_{APPLES}, q, 1000$);
    // Run pplacer on the area of T around the placement of q;
    $T'_{pplacer} \leftarrow$ pplacer($T_i$);
    // Add the location of the query sequence $q$ to a copy of $T$;
    $T_{q_i} \leftarrow$ modifyTree($T, T_i, q$);
    // RAxMLScorer runs RAxML in fixed tree mode.;
    // The output score is the maximum likelihood found on the
      tree.;
    $S_i \leftarrow$ RAxMLScorer($T_{q_i}$);
endfor
// Do a maxLoc reduction for the tree;
$bestTreeIndex \leftarrow \text{argmax}_i(S_{apples}, S_1, \ldots, S_n)$;
return $T'_{q_{bestTreeIndex}}$;
```

```
# code from Vlad Smirnov. Generates a clade of a maximum size given a leaf
def getRegion(leaf, size):
    taxons = [leaf.taxon.label]
    node = leaf
    while len(taxons) < size:
        shuffledSiblingNodes = node.sibling_nodes().copy()
        for bro in shuffledSiblingNodes:
            taxons.extend(collectSubtreeTaxa(bro, size - len(taxons)))
            if len(taxons) >= size:
                return taxons
        node = node.parent_node
    return taxons
def collectSubtreeTaxa(node, numTaxa):
    if node.is_leaf():
        return [node.taxon.label]
    taxons = []
    shuffledChildNodes = node.child_nodes().copy()
    random.shuffle(shuffledChildNodes)
    for child in shuffledChildNodes:
        taxons.extend(collectSubtreeTaxa(child, numTaxa - len(taxons)))
        if len(taxons) >= numTaxa:
            return taxons
    return taxons
```