Taxonomy Tree Alignment

UJJWAL GOEL
Center of Bioinformatics and Computational Biology
ujjwal@cs.umd.edu
Hector Corrada Bravo

Abstract

A common use case in Bioinformatics is given a set of samples from an experiment, a taxonomy tree is produced depending on the similarity between them. There can be many such taxonomy trees produced from different sources using different experimental methods. We attempt to study the similarities/differences between these taxonomy based graph databases and try to align them. This alignment would eventually lead to a global merged database which contains the knowledge acquired from multiple taxonomy tree databases. This paper focuses on the study of two popular taxonomy datasets *Greengenes* and *RDP databases*. Next we propose an approach to tackle the tree alignment problem and show the statistics of our alignment algorithm.

1. Problem - Introduction

Taxonomy trees provide us with a great way to study the relationship between various types of gene sequences and understand the similarities in their behavior at different levels. They can also be used to group different samples into groups based on the information available by clustering similar samples into same/closer groups.

There are various different sources for locating taxonomy trees generated over same/similar samples. Each of these might use different methodologies, experiments to isolate gene sequences from the cultures and then perform different types of clusterings or other techniques to generate a taxonomy tree. Hence, it's possible that these databases might have significantly different taxonomic information about the same gene sequences. The problem we are trying to solve here is finding a match between these databases and merging into a common database which captures the combined wisdom of input sources. This problem, as explained in detail in later sections, faces a lot of challenges due to structural differences in the taxonomies of different databases and we believe that identifying the right set of parameters would be the key to aligning taxonomy trees efficiently and accurately.

2. DataSets

We focus here mainly on two different taxonomy graph databases, Greengenes and RDP datasets. These datasets are generated from public databases using quality filters and after cleaning the data it is ready to be used for experimental analyses. On top of the gene sequences, a taxonomy to tree algorithm is generally applied to generate a taxonomy tree based on the similarity between different sequences, thus assigning these to different groups. This process enables us to understand and study the evolutionary behavior between different samples. Next we discuss a few details about each of the

datasets:

Greengenes [1] [2] – It focuses on Archaea and Bacteria. It comprises of two important datasets one containing a mapping of taxonomy path for an organism vs the unique id representing the organism, and second containing the gene sequences in FASTA format. The total size of this dataset, decompressed, comes to around 1.9 GB on disk and contains 1,262,986 sequences. The path for a given sequence in this database has a maximum depth of 7 based on various classes of taxonomies.

RDP [3] – It has different datasets for Archaea and Bacteria. Each contains a single file in FASTA format containing the taxonomy path as well as the complete sequences. Archaea dataset is smaller sizing up to 172 MB with 160,767 sequences and Bacteria dataset measures approximately 4 GB with 3,196,041 samples.

3. Challenges

The problem involves searching for each sequence in one database against the other database. This involves potentially dealing with many challenges each of which required specific attention. Here we mention the important ones and what we did to tackle these.

- Different databases might be in different formats. Before we try to align them, cleaning and standardization needed to be applied. Hence preprocessing was done to parse each dataset and organize data into same structured format. Paths also required some extra cleaning/trimming of characters (extra spaces, case differences, quotes, brackets) before searching.

- As seen in the last section, each of these datasets is quite large and searching for each sequence through such a large data would be quite time consuming, hence we needed to come up with some efficient approach. We used an approach inspired from Binary search through a formatted tree.

- There are a lot of sequences having the same taxonomy path in a given database, hence searching for another sequence in this database would require a linear scan comparison against all such sequences available at the found path. This linear scan cannot be avoided because of the inherent structure of the data and lack of exact matches due to uncertainty involved in the generation of samples from the experimental data.

- Taxonomy paths to the same sequence need not be alike in different databases. The reason can be due to approximations and methodologies adopted during the generation of taxonomic trees. This increases the complexity of our alignment algorithm multi-fold, since this means that we can't rely just on the exact search of the complete path of a sequence in a database. It's possible that a subset or superset of that path gives a better match against a search sequence. As mentioned in the next section, the search needs to be performed in the bottom-up fashion looking for the sequence.

- It's also possible that the taxonomy path is different *instead of just being a subset or superset* of the search path. This can lead to an extremely slow search algorithm since now it can be present anywhere in the whole database and an exhaustive search of the entire database for each search sequence is computationally intractable. To explain further, if the sequence is not found bottom up on the current path, backtracking would be required to start another search against another path in the tree. This means theoretically we might have to access *O(N)* nodes where N is the number of nodes in the

reference database for each search from second database. Total computation time would be *O(M * N)* (M – number of nodes in the second database) which turns out really slow.

- Another major issue we found while examining the database was that the same sequence in two databases might not be exactly identical. This could be due to experimental errors, slight mutations in different samples from same organism etc. Due to this we can't be sure if a mismatch between two sequences is because these correspond to different organisms or if there is an error in the data. So we needed to come up with an approach to define the notion of similarity between two search sequences. Here we use Edit Distance, which is defined as the minimum number of changes (edits/insertions/deletions) needed to convert one string to another. This metric computes the similarity between two strings, lower the edit distance, more similar the strings are. Identical strings would have an edit distance of zero.

Now instead of doing an exact comparison between search sequence and the sequences along a path in the reference tree database, we needed to perform an approximate search based on computing the edit distance between every possible pair and choosing the pair with minimum edit distance. Since the edit distance will always be computed even for the most dissimilar sequences we would need to provide a threshold to decide whether the similarity found refers to a match or not. This threshold parameter can be tuned to find the most optimal one. This adds to the computational complexity since finding the first good match is not good enough. We would have to find the best match with lowest edit distance. The idea of locating a *similar* sequence in a similar path makes the task difficult.

- We also found a lot of cases where the last level of the path started with 'unclassified' which was causing a mismatch. So we treated that word as an empty level. Basically, we just removed the last level of the taxonomy path if it started with 'unclassified'.

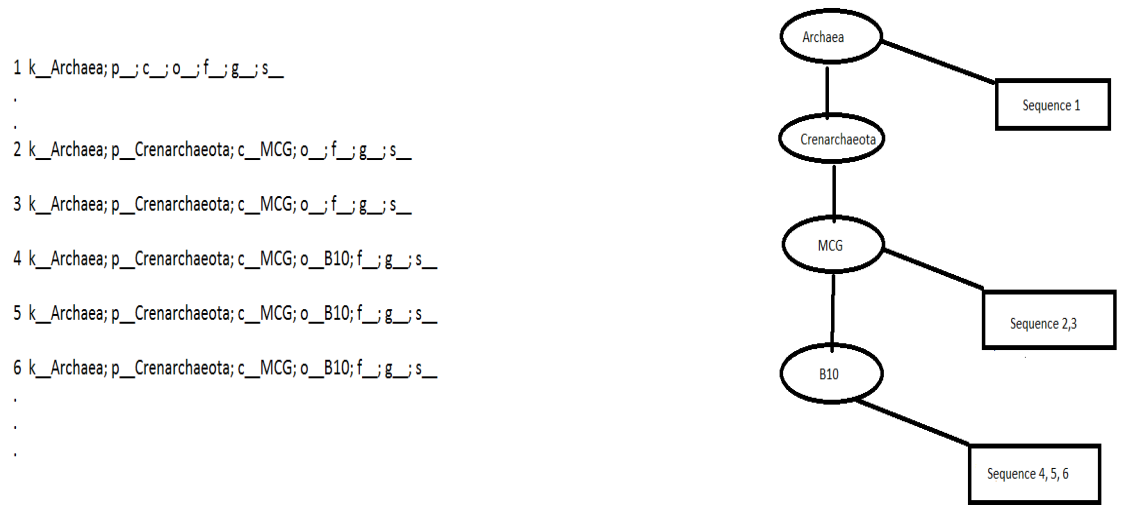

Fig 1: An example of EQ-ZONE and the corresponding taxonomy subtree.

4. Algorithm

In this section we explain the approach taken to solve the problem of taxonomy tree alignment in detail. We address each of the challenges described in previous section while designing this algorithm and try to align maximal number of sequences in the two input graph databases.

The approach taken is inspired from the Binary search algorithm. In binary search, we try to search for a given key in a *sorted list* of elements. At each step, you compare the key against the middle element of the current prospective list and based on the result of the comparison, we can eliminate one half of the list and continue the search recursively. We are going to use the similar approach to serve our purpose. The difference in this case is that there can be many possible middle entries which can match the search key instead of just one.

We introduce the notion of EQ-ZONE or Equal Zone. EQ-ZONE refers to a range of **continuous** elements in the prospect list which can possibly match the search key. This is to handle the situations where the path in one database might not be the same (but rather subset/superset) of the path in the other database. [Fig 1] shows an example of establishing an EQ-ZONE. Each path in the EQ-ZONE is either equal to the search path, a subset or a superset of the search path that we are looking to align. As we discussed in earlier section, the search sequence corresponding to the search path can be present in any of these nodes in the database. For example consider the search sequence to be "*k__Archaea; p__Crenarchaeota; c__MCG; o__; f__; g__; s__;*". Here 'Archaea' is the highest node and 'MCG' is the third and the last one in this path. Now say if we compare this against the path of sequence 3 in Fig 1 then we can see that any of the paths from sequence 2 to sequence 6 can be a potential match. Note that although sequence 1 can also be a potential match, it would not be part of same EQ-ZONE as it is not part of the continuous list.

We start by sorting the edges of each node in one of the taxonomy databases (say Greengenes) alphabetically so we can perform search to lookup for a path efficiently. Here we only sort the paths and not based on the sequences. This acts like a 'trie' data structure which enables efficient lookup for a key (path from another taxonomy database). Once we hit the leaf we can do a look up for the search sequence in the list of sequences available in this leaf node. If found, we can return with high confidence. Otherwise, we can't say for sure that the sequence is not available in the database. So we need to go up the tree along the path and check other nodes on the way. At each such node where some sequences are found, we need to perform same lookup for the sequence until either the search succeeds or completes exhaustively without a successful result. [Fig 2] shows the pseudo code for the algorithm. Since exact searches were rarely found in the dataset, we decided to modify the sequence search to minimize edit distance between two sequences and return the best match found, if this best match was within certain threshold in terms of edit distance.

In terms of run-time complexity, this search is quite time-consuming because of the structure of the problem and the inherent complexities found in common taxonomy databases (mentioned in section 3).

5. Implementation Details

The database is formatted as a sequence of paths followed by their gene sequences. We load the reference database into a dictionary where key is the path string and value refers to the actual gene

sequence. Each path in a given database generally has a limited number of levels although different databases may have different possible set of such levels. We implemented our alignment as a search of the path of each sequence in second database against the first database. Any sequence not found would be considered a new entry and 'added' to the merged database. This would ultimately lead to an alignment of two taxonomy databases and common matches besides taking the uncommon sequences from each database and adding these to merged database as well.

We model our search as a string search of taxonomy path followed by a string lookup for each gene sequence. We sort the taxonomy paths for first dataset and perform a search similar to binary search eliminating half of the dataset at each level based on the alphabetical ordering of the two paths. This runs quickly for searching a path which is extremely different from the paths in reference database and requires very few lookups (O (log N)).

After hitting a match (which includes subset or superset of search path as well) we start establishing the EQ-ZONE. For each path above and below the matched path in the current prospective range, we check if that path is a subset or superset of the matched path. We do this until we hit a path which doesn't satisfy this constraint. Now for each of these possible paths in the EQ-ZONE, we compute the search sequence against their sequence and find the best match (least edit distance) and return the sequence which gives us the lowest edit distance, if this value is below a certain threshold. If nothing is found we recursively call it on the two new regions defined above and below the EQ-ZONE. These two extra calls instead of just one in case of Binary Search makes this system more complex in terms of performance. Fig 3 shows a pseudocode of our implementation.

```
function searchTree(refTree, searchPath, searchSeq, currentMin = Infinity):
      pointer = refTree
      foreach node in searchPath:
            if pointer.children[node] is empty:
                  exit loop
      pointer = pointer.children[node]
      foreach seq in pointer.seq:
            if editDistance(searchSeq, seq) < currentMin:
                  currentMin = editDistance(searchSeq, seq)
      if currentMin < threshold:
            return true
      else:
            Backtrack and perform Depth First search on parent
            node of current pointer (bottom up)
```

Fig 2: Pseudo code for Binary Search-Align in a taxonomy Tree

5.1 EQ-ZONE

Establishing EQ-ZONE required handling several cases. Two paths A and B can be a probable match, if:

- Path A matches path B exactly.
- Path A is a substring of path B or,
- Path B is a substring of path A.

We keep expanding the EQ-ZONE as long as at least one condition is true, on either side of the middle point of the current list. This needs to be done because of the fact that the same sequence might have different paths in different databases. An important point to note is that EQ-ZONE always refers to a continuous set of paths from the database. Any path which is a substring of current search path but not continuous is taken care of in further recursive calls.

```
function searchTaxDataSet(refDataSet, start, end, searchPath, searchSeq):

    if start > end:
        return -1
    if start < 0 or end >= length(refDataSet):
        return -1

    result = compare(refDataSet, searchPath, searchSeq, start, end)
    mid = (start + end) / 2

    if searchPath lies in first half of refDataSet:
        return searchTaxDataSet(refDataSet, start, mid - 1, searchPath, searchSeq)
    else if searchPath lies in second half of refDataSet:
        return searchTaxDataSet(refDataSet, mid + 1, end, searchPath, searchSeq)
    else if a unique match is found:
        return the unique match
    else if an EQ-ZONE has been found:
        low =searchTaxDataSet(refDataSet, start, result.low, searchPath, searchSeq)
        high =searchTaxDataSet(refDataSet, result.high, end, searchPath, searchSeq)
        return the better match out of low and high
    return -1


function compare(refDataSet, searchPath, searchSeq, start, end):

    mid = (start + end) / 2
    path = refDataSet[mid]
    i = 0
    minEd = infinity

    ed = editDistance(searchSeq, path.seq)
    loop:
        nodeSearch = searchPath[i]
        nodePath = path[i]
        if nodeSearch is empty or nodePath is empty or ((i==6) and (nodeSearch ==
nodePath)):
            if ed < minEd:
                minEd = ed
            #EQ-ZONE expansion
            minEd = min(minEd, expandEQZoneAndFindMinimum(refDataSet, searchSeq,
start, end))

        if minEd < threshold:
            return best match found

        return no match found

    else if editDistance(searchPath, path) < threshold:
        minEd = min(minEd, expandEQZoneAndFindMinimum(refDataSet, searchSeq, start,
end))
```

```
    if nodeSearch < nodePath:
        return first half of the list as prospect
    elif nodeSearch > nodePath:
        return second half of the list as prospect

    i = i + 1


function expandEQZoneAndFindMinimum(refDataSet, searchSeq, start, end):

    minEd = Infinity
    mid = (start + end) / 2
    low = mid - 1
    while low >= start:
        lowPath = refDataSet[low]
        if lowPath is not a subset of searchPath:
            exit loop
        ed = editdistance(searchSeq, lowPath.sequence)
        if ed < minEd:
            ed = minEd
        low = low - 1

    high = mid + 1
    while high <= end:
        highPath = refDataSet[high]
        if highPath is not a subset of searchPath:
            exit loop
        ed = editdistance(searchSeq, highPath.sequence)
        if ed < minEd:
            ed = minEd
        high = high + 1
    return minEd
```

Fig 3: Pseudo code for Binary Search-Align as per implementation using String searching

6. Heuristic Approaches

On running our implementation we noticed that our system was running way too slow. As explained above in different sections, different variables add to the complexity making the entire problem computationally intractable. Hence we decided to try some heuristic approaches to see if we can get faster results without compromising on the accuracy of the results. In this section we discuss a few heuristics used in our implementation :

- We noticed that higher levels of taxonomy define the certainty of a sequence, while lower levels just add more information without adding much variability. Hence we decided to cutoff the search at level 5 instead of going to level 7 for Greengenes.

- We observed that the number of exact matches for the gene sequences in the database were almost non-existent. We suspect this could be due to the fact that these sequences came from different databases which were generated from different experiments. There is always a certain probability of error in the experiment plus genetic mutations etc which can cause certain differences in the same gene sequence across the databases. Hence we decided to choose the threshold for similarity as 150

(maximum edit distance allowed for a match).

- Cases were found where same sequence was available at different paths in the two databases. This was due to the lack of standardization in heirarchy across the databases. For example, RDP database has extra intermediate levels like *subclass* and *suborder* which are not present in Greengenes dataset. Again, we used the idea of edit distance to find an approximate match between two paths using the cutoff of 50.

- Instead of finding the minimum edit distance throughout the EQ-ZONE and possibly even beyond, we cut off the search as long as first editDistance less than threshold has been found.

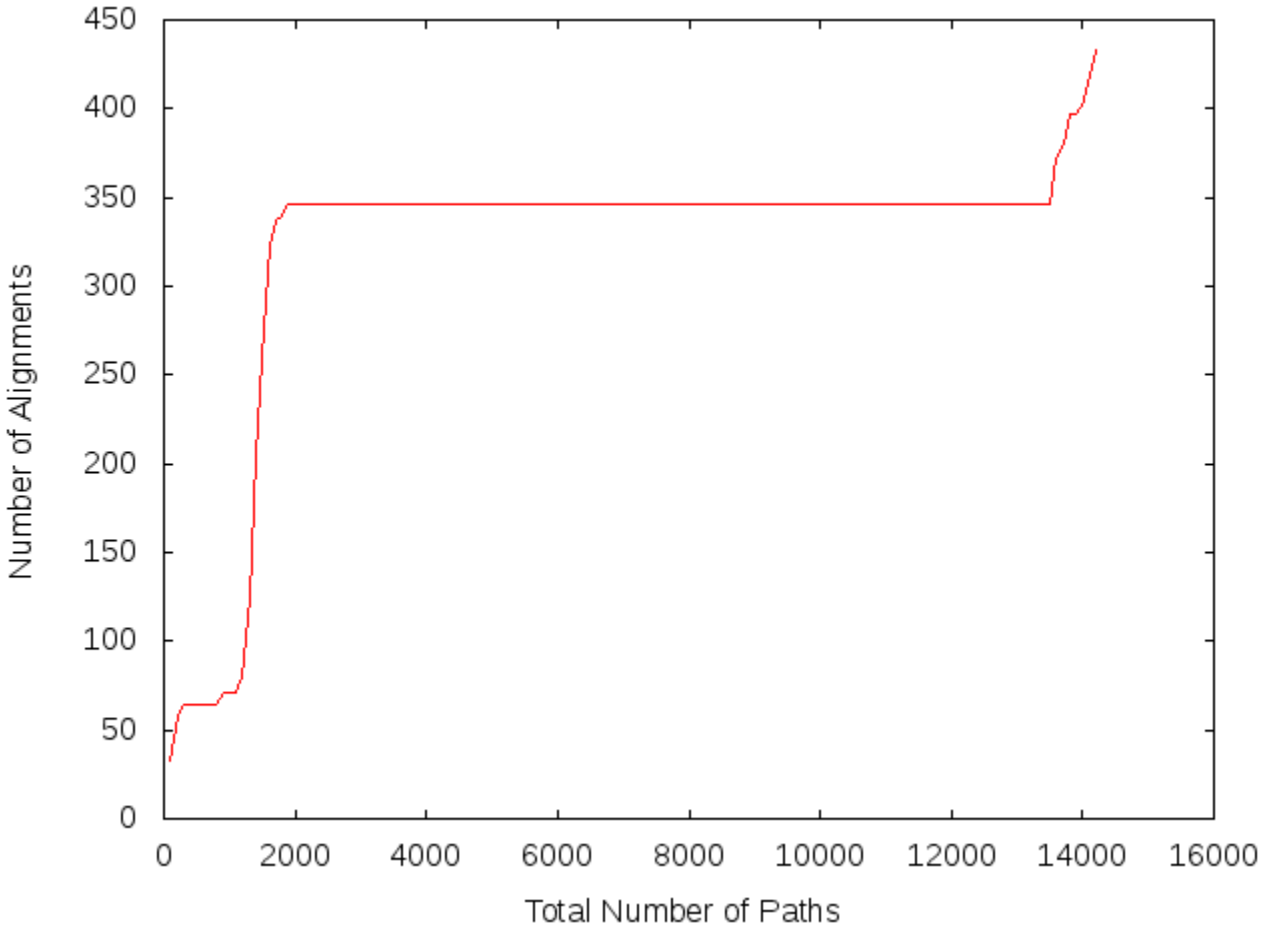

Fig 4 : Plot showing number of alignments between GG and RDP databases for Archaea as the number of sequences are scanned. There is a flat region encountered which indicates the section with clear dissimilarity between the two databases.

7. Results

The machine used to test this implementation was an Intel i7 4th generation processor (8 cores), 8 GB of RAM with 4 MB cache. The environment used to implement was python 2.7 in Ubuntu distribution of Linux environment. We focused mainly on aligning Greengenes database against RDP

database. We sorted the paths in the Greengenes dataset first making it our reference database and searched for each sequence in RDP database against this sorted dataset eventually either aligning or adding to the global database.

The sequences in Greengenes dataset count to 1,262,986 which includes both Archaea and Bacteria samples. As a test of our algorithm, we attempt to match it against the Archaea samples in RDP database. Few interesting points of observation :

- We get a match of 796 out of 160,767 (0.5%) which seems like a really low match percentage.
- We see local property during alignment. There are sections which match in bunch and then no matches happen for a group of sequences. This indicates that there are subtrees in one database which are being aligned successfully to corresponding subtrees in the second database.
- As shown in Fig 4, there are sections of sequences in RDP where no matches happen for a very long stretch. These sections seem to be drastically different from any sections in Greengenes.
- The time taken by alignment for any section is proportional to the number of matches happened in that section. What this means is that matches take a long time since this needs to establish an EQ-ZONE and search through a larger chunk of database to ascertain that it's a valid alignment while mismatches can be quickly determined based on the logic similar to binary search algorithm.
- Results are identical for exhaustive vs heuristic search approaches.

7. Future Work

This problem needs further study to improve the performance and accuracy of matching. The threshold parameters can be tuned to find an optimal value. We need to further investigate the reason for stagnancy in the middle which is causing so many mismatches and hence the overall low match percentage. Another approach would be to explore other algorithms which can take advantage of the locality of matching data within a database. For example, we don't need to do an exhaustive search throughout the database. If it's not found in the near neighborhood of its neighbor's match, it's highly unlikely that the node is present in the reference database. We can also look at ways to scale our solution by making it distributed and running it on a cluster.

8. References

[1] McDonald D, Price MN, Goodrich J, Nawrocki EP, DeSantis TZ, Probst A, Andersen GL, Knight R and Hugenholtz P. (2011). An improved Greengenes taxonomy with explicit ranks for ecological and evolutionary analyses of bacteria and archaea. The ISME Journal 6:610-618. doi:10.1038/ismej.2011.139.

[2] DeSantis, T. Z., P. Hugenholtz, N. Larsen, M. Rojas, E. L. Brodie, K. Keller, T. Huber, D. Dalevi, P. Hu, and G. L. Andersen (2006), Greengenes, a chimera-checked 16S rRNA gene database and workbench compatible with ARB, Appl Environ Microbiol, 72(7), 5069-5072.

[3] Cole, J. R., Q. Wang, J. A. Fish, B. Chai, D. M. McGarrell, Y. Sun, C. T. Brown, A. Porras-Alfaro, C. R. Kuske, and J. M. Tiedje. 2014. Ribosomal Database Project: data and tools for high throughput rRNA analysis Nucl. Acids Res. 42(Database issue):D633-D642; doi:10.1093/nar/gkt1244 [PMID: 24288368]

[4] Quast C, Pruesse E, Yilmaz P, Gerken J, Schweer T, Yarza P, Peplies J, Glöckner FO (2013) The SILVA ribosomal RNA gene database project: improved data processing and web-based tools.*Nucl. Acids Res.41 (D1): D590-D596.*

[5] Yilmaz P, Parfrey LW, Yarza P, Gerken J, Pruesse E, Quast C, Schweer T, Peplies J, Ludwig W, Glöckner FO (2014) The SILVA and "All-species Living Tree Project (LTP)" taxonomic frameworks. *Nucl. Acids Res. 42:D643-D648*