Large Scale and Dynamic Phylogenetic Inference from Epidemic Data

Marta Alexandra Fragoso Nascimento
marta.nascimento@tecnico.ulisboa.pt

IST, University of Lisbon, Portugal

October 2017

Abstract

Typing methods are widely used in the surveillance of infectious diseases, outbreak investigation and studies of the natural history of an infection. Their use is becoming standard, in particular with the introduction of High Throughput Sequencing (HTS). On the other hand, the data being generated is massive and many algorithms have been proposed for phylogenetic analysis of typing data, addressing both correctness and scalability issues, such as the goeBURST algorithm. Most of the distance based algorithms for inferring phylogenetic trees follow the closest-pair joining scheme. This is one of the approaches used in hierarchical clustering. Although phylogenetic inference algorithms may seem rather different, the main difference among them resides on how one defines cluster proximity and on which optimization criterion is used. The main goal of this thesis is to study the most well known phylogenetic inference methods suitable for processing typing data, focusing on the goeBURST algorithm and the computational problems that appear when dealing with large datasets. This algorithm must however be run whenever new data becomes available starting from scratch. We address this issue by proposing two dynamic algorithms allowing data to be continuously integrated and updated. Experimental results show that these algorithms are efficient at integrating new data and updating inferred evolutionary patterns, improving the update running time by at least one order of magnitude.

Keywords: phylogenetic inference; phylogenetic trees; dynamic algorithms; sequence-based typing data

1. Introduction

Biological sequences, namely DNA (deoxyribonucleic acid), RNA (ribonucleic acid) and proteins, have a central role in molecular biology because they define almost every cellular activity that occur in the each organism. The key to uncover these processes relies in understanding how these sequences interact with each other in their own environment. In order to do that, it is necessary to sequence the DNA and identify the genes that are part of the genome. For this, a database with known genes for that specific organism is used and the matching is made based on some specific allele according to the chosen typing method. This way, and by the comparison between different genes, it is now possible to identify its lineage, and then infer evolutionary relationships through phylogenetic analysis.

The choice of which typing method to use depends on the epidemiological context. These methods provide further knowledge on surveillance of infectious diseases, outbreak investigation and the natural history of an infection. Furthermore, with the introduction of "High Throughput Sequencing" (HTS) [1] technology, there have been developed new typing methods likeribossomal-Multilocus Sequence Typing (ribossomal-MLST) [2], Multilocus Variable Number of Tandem Repeats Analysis (MLVA) [3] or analysis of Single Nucleotide Polymorphism (SNPs) [4] through comparison against a reference genome. Moreover, these recent advances and the resulting decrease in costs that allow the analysis of thousands of data, created the need of developing new efficient methods that are able to do phylogenetic analysis and that also allow the dynamic updating of data into an ongoing study.

The number of practical applications of phylogenetic analysis continues to grow and are by no means limited to the functional and structural study of genomes. Phylogenetic applications span much of biology, from human health [5] and forensics [6, 7] to conservation biology [8] and studies of behavior [9]. Many of these applications require accurate phylogenetic estimates, not only in terms of tree topologies, but also in branch lengths (for estimation of time and/or the amount of change), ancestral character states (for estimation of evolu-

tionary transitions), and parameters of evolutionary models (for study of evolutionary processes). Therefore, phylogenetic analysis has become central to understanding biodiversity, evolution, ecology, and genomes.

1.1. Objectives

The aim of this thesis is to study some well-known algorithms for phylogenetic inference for processing large scale epidemic data. These methods have been proposed before the actual massive growth of epidemic data and, hence, most of existing implementations are facing great challenges with existing datasets. On the other hand, these methods rely often on graph theory and network mining ideas, which have been evolving in recent years, namely in what concerns dynamic graphs. Also, if we are dealing with missing data in large-scale phylogenomic datasets we may want to try different predictions since making the wrong ones will lead to negative effects on the phylogenetic inference process [10]. Hence, the challenge is to apply some techniques on two of those algorithms for phylogenetic inference, namely the distance-based-MST and goeBURST Full MST algorithms [11], allowing data to be continuously integrated and updated without requiring us to compute everything from start (*i.e.* dynamically update metrics and phylogenetic trees). Hence, it is expected that they construct the same phylogentic trees.

This work and related implementations will rely on, and extend the PHYLOViZ framework [12, 13].

2. Background

Phylogenetic analysis aims at uncovering the evolutionary relationships between different species (OTU – Operational Taxonomic Units), to obtain an understanding of their evolution. Phylogenetic trees are widely used to address this task and are reconstructed by several different algorithms [14]. However, a phylogenetic tree will not always suffice to correctly represent the evolutionary history of a population and sometimes a network representation will be more appropriate [15]. Phylogenetic networks provide an alternative to phylogenetic trees and may be more suitable for datasets whose evolution involves significant amounts of reticulate events caused by hybridization, horizontal gene transfer, recombination, gene conversion or gene duplication and loss [16]. However, they are hard to analyze and thus phylogenetic trees are more common. Therefore, we focus our work on phylogenetic trees and on the algorithms that are commonly used to reconstruct them, namely on distance-based algorithms, where the Hamming distances between pairs of sequences are computed [17].

2.1. Distance Matrix Methods

Distance matrix methods take as input the pairwise distances for a set of OTUs, represented by a matrix D. These distances do not take into consideration the number of back mutations[1] and multiple mutations that occurred at the same position and therefore it underestimates the true evolutionary distance. To rectify this, a correction formula based on a model of evolution is often used [16]. For example, in the case of the Jukes-Cantor model, it assumes all substitutions are independent, equal base frequencies (*i.e.* all sequence positions are equally subject to change), equal mutation rates and no insertions or deletions have occurred. Given the proportion of positions that differ between two sequences A and B (*i.e.* hamming distance), $H(A, B)$, the Jukes-Cantor estimate of the evolutionary distance (in terms of the expected number of changes) between two sequences is given by Equation (1).

$$JC(A, B) = -\frac{3}{4}\ln(1 - \frac{4}{3}H(A, B)) \qquad (1)$$

So, if we assume that those sequences evolved according to the Jukes-Cantor model of evolution, to compute a distance matrix that approximates the true evolutionary distances, we first determine the Hamming distance between any two sequences A and B and then apply this transformation to get a corrected value.

The actual tree is then computed from the distance matrix, possibly corrected according to some evolution model, by running a clustering algorithm that starts with the most similar sequences (Globally Closest Pair) or by trying to minimize the total length of the tree (Minimum Evolution).

All these methods can be generalized by algorithm 1.

2.1.1 Globally Closest Pairs methods

GCP based algorithms are widely used in phylogeny. They receive as input a dissimilarity matrix containing all pairwise differences between elements and return a hierarchy of clusters.

First, a pair of clusters is chosen based on the minimum dissimilarity criterion over the matrix D. If a tie occurs, the selection between those cluster-pairs is arbitrary. Then, the selected pair (C_i, C_j) is removed from the set, joined together to form one single cluster ($C_u = \{C_i \cup C_j\}$) and added to the hierarchy $\mathcal{H}$. The distance between each element of the selected pair to the new cluster (D_{iu} and D_{ju}) is $D_{ij}/2$. Finally, in the reduction step 3, all dissimilarities from C_i and C_j to any other element need to be recalculated taking into account the new cluster

[1] back mutation is a mutation that restores the original sequence and hence the original phenotype.

Algorithm 1: General scheme for hierarchical agglomerative clustering methods based on distance matrices.

Input: A matrix D over a set of elements S (OTUs).

Output: A cluster-hierarchy $\mathcal{H}$ over S.

Initialization: Initialize the cluster-set C by defining a singleton cluster $C_i = \{i\}$ for every element $i \in S$. Initialize output hierarchy $\mathcal{H} \leftarrow C$.

Loop: While $|C| > 1$ do:

1. **Cluster-pair selection:** Select a pair of distinct clusters $\{C_i, C_j\} \subseteq C$ from D, according to some criterion.
2. **Cluster-pair joining:** Remove C_i, C_j from the cluster set C and replace them with $C_u = \{C_i \cup C_j\}$. Add C_u to the hierarchy $\mathcal{H}$ and calculate the branch length for each element (D_{iu} and D_{ju}).
3. **Reduction:** Update matrix D by calculating the new values (C_{uk}) for every $C_k \in C' \setminus \{C_i \cup C_j\}$.

Finalize: Return the hierarchy $\mathcal{H}$.

C_u in order to update the dissimilarity matrix. The algorithm ends when there are no more clusters to join.

UPGMA (Unweighted Pair Group Method with Arithmetic-mean), WPGMA (Weighted Pair Group Method with Arithmetic-mean), SL (Single-Linkage) and CL (Complete-Linkage) are different variants of GCP, and they all differ on the reduction formula used in step 3. UPGMA defines the similarity between two clusters as their average dissimilarity. WPGMA defines it by weighting the two clusters by 1/2. Single-linkage chooses the minimum value and complete-linkage the maximum.

2.1.2 Minimum Evolution principle

Clustering methods try to find the optimal tree using different evolution models. Some models use the minimum evolution principle, where a tree is considered to be optimal if it has the shortest total branch lengths. In order to get the minimum evolution tree it must be decided how the branch lengths are estimated and then how the tree length is calculated from these branch lengths.

The neighbor-joining is the most common method and has been widely used in phylogeny inference. All variants of this method try to improve on its accuracy in reconstructing the true phylogenetic tree and also by trying to decrease its computational cost. Here we describe and present the most widely used variants of this method.

2.1.2.1 Neighbor-Joining and variants

Neighbor-Joining algorithm is the most commonly used method in phylogenetics and several variants of this algorithm have been introduced over the years. While some try to optimize the formulas used by NJ to better estimate the true and optimal tree others try to improve NJ's efficiency, both in terms of running time and memory usage.

NJ (by Studier and Kepler [18]), UNJ, BIONJ, FNJ and ClearCut methods are variants of NJ (by Saitou and Nei [19]) that differ from the latter by applying different criteria over the three steps of the generalized algorithm 1. These algorithms take as input a dissimilarity matrix D and then create a new matrix Q based on the application of some criterion over D; Q is then used in the step 1 of algorithm 1 as criterion, being updated along D. This criterion relies on the minimum evolution principle. Branch lengths are computed in step 2. Finally, in step 3, the dissimilarity matrix is reduced by deleting a pair of OTUs (i, j) and by estimating new distances between the new OTU u to any other OTU k (D_{uk}).

Regarding neighbor-joining efficiency, several methods were developed to address running time and memory issues [20, 21, 22, 23, 24, 25, 26]. Nevertheless, and regardless of the implemented optimization technique, their ultimate goal is always trying to obtain the most accurate tree efficiently.

2.1.3 Distance-based-MST-like methods

As we have seen earlier, the theory of evolution predicts that existing biological species have been linked in the past by common ancestors and their relationships can be depicted as a branched diagram like phylogenetic trees. The methods that we will present here use the same evolutionary model of NJ (minimum evolution) to better estimate the true phylogenies however they were developed following a graph theoretic approach.

2.1.3.1 Generic-MST

The problem of finding a minimum spanning tree can be defined as growing the minimum spanning tree one edge at a time. At each step, we determine an edge (u, v) that we can add to $\mathcal{T}$ without creating a cycle in it. Therefore, all edges added to $\mathcal{T}$ guarantee that $\mathcal{T}$ must be a minimum spanning tree. In the case of a tie, *i.e.*, if a new vertex u has two safe edges, (u, v) and (u, w), connecting to the tree $\mathcal{T}$ with equal weights, we must define a comparator, *cmp*, to decide which one to choose.

There exist several algorithms that elaborate on this generic method and they use a specific criterion to determine a safe edge (*e.g.* Kruskal, Prim, Borůvka, distance-based-MST, ...).

2.1.3.2 L.R. Foulds, M.D. Hendy and David Penny

This proposed method is an heuristic method of approaching the phylogenetic problem. At each iteration, the method identifies the shortest link joining two unconnected points, which when added individually to the existing graph do not create a cycle. Then, it tries to reduce the graph by "coalescing", where necessary. This means that after a link is selected and added to the tree it compares all the pairs of links incident there, and identifies the changes that are common to the two links, choosing to coalesce on a pair with the maximal number of common changes. If a cycle is present an attempt is made to break it by removing links which give the largest reduction in the total length of the tree. If there is more than one largest link, the cycle is left unbroken because it can subsequently be broken by later additions. The method terminates when all original species are connected. Although, if at the end the cycles are still unresolved, all the alternative trees are presented and the choice is left to the biologist who may wish to use some additional information.

2.1.3.3 goeBURST

goeBURST algorithm [28] is a globally optimized implementation of the eBURST algorithm [29] that identifies alternative patterns of descent for several bacterial species.

This algorithm can be stated as finding the maximum weight forest or, depending on weight definition, as finding the minimum spanning tree. Therefore, the optimal solution can be provided by a greedy approach on identifying the optimal forest with respect to the defined partial order on the set of links between STs. Due to its desirable properties and ease of implementation, goeBURST uses Kruskal algorithm to achieve that goal.

It consists of building a spanning forest in a graph where each OTU is a node and two OTUs are connected if and only if they are at distance one (*i.e.* if they only are SLVs). Since this forest should be optimal with respect to link selection, the links between OTUs with higher number of SLVs must be selected. In case of a tie it should be considered the number of OTUs that are at distance two (*i.e.* number of DLVs), the number of OTUs that are at distance three (*i.e.* number of TLVs), the occurrence frequency of OTUs, and lastly the assigned OTU number (ID). Although this last tiebreak is rarely reached, this criterion is necessary to provide a consistent and unique solution to the problem as it will always provide a consistent tiebreak solution due to the uniqueness of the ID. Lower IDs take precedence over higher IDs because it is assumed that in a growing database with data of several contributing international studies, the more common OTUs are sampled first and will have lower ID than the subsequent studies that will add more OTUs to the database. These rules define a partial order on the set of edges between vertices [28] and therefore goeBURST can be considered a distance-based-MST algorithm.

The goeBURST scheme is very similar to algorithm 1 but it skips the reduction step. The final step is not necessary because it does not need to update the overall pairwise distances. It also provides an optimal solution for the link assignment, since it performs a global optimization taking into consideration all possible ties at all levels between STs in the data set.

2.1.3.4 goeBURST Full MST

Using an extension of the goeBURST rules defined above up to nLV level (where n equals to the number of loci in a strain), a Minimum Spanning Tree-like structure can be computed. If one define n as one, two or three, the results of this algorithm will be equivalent to calculating goeBURST at those levels (SLV, DLV and TLV respectively).

For the full MST computation the Borůvka algorithm is used to build the tree, which is also a greedy algorithm for finding a minimum spanning tree in a graph but with the constraint of having all edge weights distinct [30, 31].

3. Dynamic distance based MST algorithms

The MSF problem in goeBURST and the MST problem in generic-distance-based-MST and goeBURST Full MST are particular cases of graphic matroids [32]. Thus, finding a solution to represent the phylogenies by the use of MSTs consists of solving instances of graphic matroids [32, 33, 34] which can be optimally solved with a greedy approach [35].

Thus, given a graph G and the set $\mathcal{F}$ of all forests over G (*i.e.* a matroid), the optimization problem is to find an optimal forest. Note that by allowing any possible distance for link comparison and thereby fully generalizing link comparison, any problem of tree construction becomes a graphical matroid instance [36, 37], that can be solved by several greedy algorithm, such as the Borůvka algorithm [30], the Kruskal algorithm [38] or the Prim algorithm [39].

3.1. goeBURST Full MST

This static version of goeBURST Full MST starts by defining the maximum number of loci under analysis (*i.e.* maximum level) and then we compute for each OTU how many other OTUs are at distance one, two, etc. In other words, we compute the total number of SLVs, DLVs, TLVs and so on, till we reach the defined nLV level, for every single OTU. Finally, we run an adapted version of Borůvka's algorithm where the merging phase occurs only between components at the same level (*i.e.* equal weights).

The main difference between Borůvka's algorithm and its adapted version relies on the definition of cheapest edge. This means that instead of selecting the minimum weight edge for each OTU it starts by selecting an edge that is at distance one from that OTU. This is done iteratively for each level till there are no more edges at that level. Once more, if there exists more than one edge for the same level, the goeBURST tiebreak rules are applied till no edges for that level remain.

3.2. Dynamic MST algorithms

The goal here is allow the addition of a new OTU to a previously computed minimum spanning tree without the need of running a static distance-based-MST algorithm from scratch. This previous computed tree is stored in secondary memory, hence our approaches take that into consideration.

The adding operation can be divided mainly into two major steps: pre-processing and adding the new OTU. The pre-processing step is common to all algorithms and is responsible for creating all structures that are necessary for the addition step, while the latter refers to the specific process of adding the new OTU. Here, we will detail the adding step for each algorithm.

3.2.1 Dynamic generic distance based MST algorithm

This dynamic algorithm is the simplest because it is only based on the pairwise distances. Given a new OTU, u, it starts by adding the first new edge, $(u, v) \in E'$, directly in the MST. Since we are adding u for the first time in the tree, it will not be necessary to check if it would form a cycle in the tree or not. Then, iteratively, we add each remaining new edge (in increasing order of the distance) and check if it will create a cycle by traversing the tree with a BFS (breadth-first search) approach. If that occurs and if the new edge weight (distance) is less than the weight of the highest weight edge in this cycle, then we create a lower weight MST by replacing that higher weight edge with the new edge. Otherwise, the current MST remains optimal and we discard that new edge.

3.2.2 Dynamic goeBURST Full MST

The addition step regarding dynamic goeBURST Full MST algorithm is more complex than the latter because it can change the entire tree. This is due to the fact that the overall number of locus variants for all OTUs will vary causing a possible change on the tiebreak rule applied before by goeBURST Full MST algorithm. Therefore, we need to start by updating those values for each OTU in the tree according to the distances to the new OTU, and sort them according to the specified comparator. This comparator is based on the tiebreak rules defined by goeBURST.

Then, iteratively and by distance level, we start adding the sorted edges to the new MST, T'. As we do not know if the tiebreak applied to the remaining edges has changed we also need to check and compare all edges $(u, v) \in E$ in the original MST, T, for that same level, and also all relevant edges of u and v. We define (u, v') as a relevant edge of u if $D_{uv} = D_{uv'}$ and $v \neq v'$, meaning that the algorithm that generate that MST had to use a tiebreaker to decide which edge to add.

4. Implementation

Here, we reason about our solutions for the proposed algorithms and discuss the implementation details for each one. We will start by briefly introducing part of the framework that gives support to these algorithms and then we describe the implementation of each one of them.

4.1. Framework

goeBURST Full MST algorithm is classified as distance matrix method and therefore in order to implement it as we described in the last chapter we need to have a matrix as input. This matrix can be provided directly or generated from allelic profiles. This means that the pairwise dissimilarity can be obtained either by a distance vector that represents directly the distance from one element to the others or through several profile comparisons using some similarity measure. To allow these two types of input we created a generic interface called `PairwiseDissimilarity<T>`, where `T` corresponds to a specific `Identifiable`. This type is represented by an interface and can correspond either to a `DistanceVector` or to a `Profile`.

Now that we have obtained all distances regardless of the type of input provided, we can now build the final matrix that will support our algorithm. As one may recall from algorithm 1, namely step 2 and 3, this matrix needs to be dynamic in the sense that it must allow the insertion and removal of some of

its elements and also allow distances to be updated. This structure was already implemented efficiently in [13].

These algorithms were developed in *Java* and they are available in https://gitlab.com/martanascimento/dynamic-phyloviz/wikis/home.

4.2. Static goeBURST Full MST

This algorithm receives as input a matrix M, explained earlier, an integer value representing the maximum number of loci that we want to analyze and a graph G that represents the adjacency lists of all OTUs in the matrix M. Then, iteratively by distance level, it iterates over all OTUs to obtain all edges that are at that distance. Then, for each edge $(u, v) \in E$ if u and v are in different trees, we add that edge (u, v) to a new set E'. This new set must be an ordered set since it will have multiple edges with the same distance that need to be prioritized according to a set of tiebreak rules. To efficiently sort those edges we use a self-balancing binary search tree, more specifically a red-black tree (`java.util.TreeSet`), that uses a comparator based on those rules. Then, for each edge (u, v) in E' if u and v are in different trees, we add (u, v) to the MST $\mathcal{T}$. To trace the OTUs in each tree efficiently we use disjoint sets data structure [31]. This process repeats till all OTUs are in the final MST or the maximum level is reached.

4.3. Dynamic MST algorithms

These dynamic algorithms allow the addition of a new OTU to a previously computed minimum spanning tree, T, without the need of running goeBURST Full MST algorithm from scratch. In order to do this, and for now, we will only need that previous MST, T. Here we will explain how we implemented the adding steps for each dynamic algorithm.

4.3.1 Dynamic generic distance based MST algorithm

The process of adding a new OTU to a MST is simple because we only have to detect if a cycle is present or not. This can be done using a queue (`java.util.Queue`) to store all OTUs (neighbors) that were still not visited by BFS, an array of boolean values (`boolean[]`) to mark each OTU as visited or not and an array of integers (`int[]`) to store the parent ID of each OTU. This last array will be very useful to form the path soon as we find the cycle.

Suppose that we add an edge $(u, v) \in E'$ to the tree. We start by defining u as the root of our tree and then we look for v by going through all neighbors of u that were not visited yet and enqueuing them. These neighbors can be found efficiently using the graph G data structure that represents the adjacency lists of all OTUs. If we cannot find v, then we restart this process for every enqueued neighbor, otherwise, we backtrace the all process through the parents' array adding each to the cycle path.

The adding stage for this dynamic algorithm only depends on the time took to detect the presence of a cycle. For that reason it takes linear time in the size of the graph to add a new OTU.

4.3.2 Dynamic goeBURST Full MST

The implementation of this algorithm is a little bit more complex than the generic-distance-based-MST one. It receives as input an ordered list of edges E' and a graph G that were created in the pre-processing step, a maximum distance level and also, a `MSTResult`, R. Till this point, we have only needed from a previous computation a MST. However, we have mentioned that this algorithm also needs to know how this tree was created, *i.e.*, for each level the subtrees that were clustered. Therefore, to represent each subtree, we created a new data structure called `MSTCluster` that contains a list of edges. Not only the ones that represent it but also all edges that were deferred at each level. So, the `MSTResult` data structure that we receive as input contains not only the minimum spanning tree as a collection of edges but also a collection of clusters.

The algorithm starts by updating the overall number of locus variants for each OTU. Then, iteratively and by distance level ℓ, selects all edges from all clusters that are at distance level ℓ and all new edges from E' that are at the same distance and adds them on a new set E''. For each edge $(u, v) \in E''$, we update the clusters for u and v with all edges that are at distance ℓ from u and v, respectively, and if it connects different subtrees in $\mathcal{T}'$ we add it in the new tree $\mathcal{T}'$ and we merge the previous two clusters into one.

The adding stage for this dynamic algorithm depends on the time took to get all edges for each level, the time to sort them and also the time took to create and update the clusters. For all these reasons this algorithm takes a quasilinear time in the worst case, $O(n \log n)$ in the size of the graph to add a new OTU in a minimum spanning tree.

4.4. Data persistence

After running all the proposed algorithms they output a phylogenetic tree in newick format. However, goeBURST Full MST and dynamic goeBURST Full MST algorithms also output a `.result` file that

corresponds to the serialization of the *Java Object* `MSTResult`. This file can then be used by any dynamic algorithm, through deserialization, to allow the addition of a new element to it.

This solution for loading data can be discussed since dynamic generic-distance-based-MST algorithm may not need the information of the entire tree and consequently only loading into memory the information regarding the path found by breadth-first-search.

5. Experimental Evaluation

To evaluate the performance of the dynamic algorithms, we ran them against our implementation of goeBURST Full MST where we can observe the efficiency of our solutions. We performed the tests for the *Streptococcus pneumoniae* MLST dataset available in PubMLST[2], with the number of OTUs varying from $n = 10$ to $n = 1000$ and running times averaged over 500 executions. We also used two Single Nucleotide Polymorphism (SNP) [40] datasets, one with random data and *Salonella typhi*, with the number of OTUs varying from $n = 10$ to $n = 400$ and $n = 10$ to $n = 1000$, respectively, and running times averaged over 100 executions. However, we will only present here the results regarding MLST dataset.

All experiments were performed on a machine with the following specifications: Intel Core i5 2.7 GHz quad core processor with 8 GB of memory.

5.1. Time

Streptococcus pneumoniae MLST dataset contains a profile length of 7 and a total of 1000 profiles (OTUs). If we look at Figure 1, that represents a time comparison between the three different algorithms, we can observe the time, in milliseconds, that the static algorithm takes to add n OTUs against the time that both dynamic algorithms take to add a new OTU to a MST with $n - 1$ OTUs. For instance, goeBURST Full MST takes about ≈ 400 ms to add 500 OTUs, while dynamic generic-distance-based-MST and dynamic goeBURST Full MST take much less time (about ≈ 10 ms and ≈ 30 ms, respectively) to add a new OTU to a previously computed 499 OTUs MST. This clearly shows the great advantage of the dynamic algorithms on integrating new data and updating inferred evolutionary patterns.

[2] https://pubmlst.org

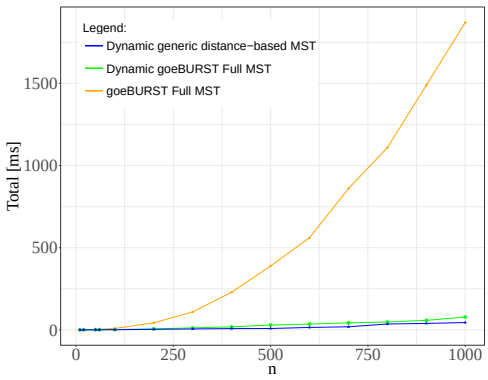

Figure 1: Dynamic vs static algorithms running time comparison as new OTUs are incrementally integrated for *Streptococcus pneumoniae* MLST dataset.

We can also make a direct comparison between both dynamic algorithms. We can observe that generic-distance-based-MST is faster than the dynamic goeBURST Full MST. This analysis supports our study since the dynamic generic-distance-based-MST process of updating a minimum spanning tree is much simpler because it only depends on the pairwise distances between OTUs. This algorithm just checks if an edge creates a cycle on the graph and, if that is the case, it removes the heaviest edge from that same cycle. However, this difference is not substantial when we compare it with goeBURST Full MST because any of them is much better than the latter on integrating new data.

5.2. Memory

The results in the Table 1 were obtained using the *Java* interface `MemoryPoolMXBean`. It represents a management interface of the memory resources managed by *Java* virtual machine. This way we can get the peak of memory usage of a memory pool since the virtual machine was started.

Both dynamic algorithms use about the same amount of resources but more when comparing to goeBURST Full MST. This is due to all the structures that they need beforehand to execute, unlike the static algorithm that starts from scratch. So, it is interesting to note that, although it is more or less the double, it grows linearly, which is acceptable since we are supporting a dynamic structure.

6. Conclusions

Large epidemiological studies on pathogen populations start to emerge as sequencing technologies become commodity, continuously generating huge volumes of typing data, and also ancillary data. And there is no doubt about the importance of such studies for the surveillance of infectious diseases and the understanding of pathogen population genetics and evolution. There are still however a number of challenges. Phylogenetic analysis is one of the main tools used in this context and, although there are many phylogenetic inference methods as we saw before, their differences and similarities are not clear most of the time. On the other hand, given the

Table 1: Memory analysis, in megabytes, of *Streptococcus pneumoniae* dataset for the proposed algorithms.

n	goeBURST Full MST	Dynamic generic-distance-based-MST	Dynamic goeBURST Full MST
10	3.25	3.90	3.90
100	4.55	5.20	5.20
200	8.45	12.35	11.05
300	13.19	28.60	25.35
400	19.82	34.25	34.25
500	28.47	35.16	34.80
600	52.11	35.31	35.30
700	51.84	68.43	68.42
800	52.27	119.07	90.58
900	73.31	137.14	137.21
1000	74.95	187.71	156.56

huge volume of ever growing data to analyze, many methods are becoming unpractical due to their computational complexity.

We provide in this paper an unifying view on most well known phylogenetic inference methods suitable for processing typing data. As we discussed, these methods share a common algorithmic background and differ only on optimization criteria and related evolution models. Taking this observations into account, one can better understand the difference among these methods and, from a computational point of view, can address simultaneously several challenges in common with all algorithms. Moreover, given that datasets are not only huge but are growing continuously, dynamic updating is becoming mandatory. And given the strict relation between clustering and these methods, we believe that well known techniques developed in last years for clustering large data can be useful in this context.

Based on the study presented here, we implemented goeBURST Full MST method and two dynamic updating techniques: dynamic generic-distance-based-MST and dynamic goeBURST Full MST algorithms. Although there are well known dynamic algorithms for computing and updating minimum spanning trees, they are not directly usable in this context due to distance updating and tie breaking rules based on locus variants.

The main contributions of this thesis are the unified view of the most commonly used inference methods and the improvements over state of the art graph clustering methods in what concerns dynamic graphs. In particular we contributed with practical and scalable implementations of the proposed methods providing always an optimal solution.

6.1. Future work

There are several possible continuations of the work done in this thesis. One could extend this study to several different algorithms. We have already provided an explication on how globally closest pairs methods can be dynamically updated, and therefore it could be a good starting point to try to apply some of the techniques that were explained in that context to other algorithms. It will also be interesting to extend this work by studying and implementing the update operation on the dynamic algorithms, instead of just the addition of a new element.

Most of the discussed hierarchical clustering methods were developed a long time ago, hence studying different ways to optimize these methods could have a major impact on their performances and, thereby, their usability.

Another interesting work would be how would different evolution models impact on the results of an algorithm in computing the phylogenetic tree and how close their results are when comparing to the real tree.

However, exploring new methods and new dynamic techniques are, in our opinion, the most interesting directions for future work.

Acknowledgements

I am using this opportunity to express my gratitude to everyone who supported me throughout the course of this Master Thesis.

I would like to express my special thanks to my supervisors Prof. Alexandre Francisco and Prof. Cátia Vaz for their insight, support and sharing of knowledge that has made this Thesis possible. The opportunity and knowledge acquired express my gratitude.

To João Carriço and fellow researchers at INESC-ID for introducing me to interesting problems in the bioinformatics' field. I learned a lot and working with them has been a great experience.

I would also like to thank to all my friends and colleagues that helped me grow as a person and were always there for me during the good and bad times in my life.

Last but not least, to my family for their friendship, encouragement and caring over all these years, for always being there for me through thick and thin and without whom this project would not be possible. Thank you.

References

[1] Jason A Reuter, Damek V Spacek, and Michael P Snyder. High-Throughput Sequencing Technologies. *Molecular Cell*, 58(4):586–597, 2015.

[2] M C J Maiden, J A Bygraves, E Feil, G Morelli, J E Russell, R Urwin, Q Zhang, J Zhou, K Zurth, and D A Caugant. Multilocus sequence typing: A portable approach to the identification of clones within populations of pathogenic microorganisms. *Proceedings of the National Academy of Sciences*, 95, 1998.

[3] Bjørn-Arne Lindstedt. Multiple-locus variable number tandem repeats analysis for genetic fingerprinting of pathogenic bacteria. *ELECTROPHORESIS*, 26(13):2567–2582, 2005.

[4] A collaborative learning space for science - SNP.

[5] R M Bush, W M Fitch, C A Bender, and N J Cox. Positive selection on the H3 hemagglutinin gene of human influenza virus A. *Molecular Biology and Evolution*, 16(11):1457–1465, 1999.

[6] David M Hillis, John P Huelsenbeck, Clifford W Cunningham, and Others. Application and accuracy of molecular phylogenies. *Science-AAAS-Weekly Paper Edition-including Guide to Scientific Information*, 264(5159):671–676, 1994.

[7] Michael L Metzker, David P Mindell, Xiao-Mei Liu, Roger G Ptak, Richard A Gibbs, and David M Hillis. Molecular evidence of HIV-1 transmission in a criminal case. *Proceedings of the National Academy of Sciences*, 99(22):14292–14297, 2002.

[8] Keith A Crandall, Olaf R P Bininda-Emonds, Georgina M Mace, and Robert K Wayne. Considering evolutionary processes in conservation biology. *Trends in Ecology & Evolution*, 15(7):290–295, jul 2000.

[9] Em\'\ilia P Martins. *Phylogenies and the comparative method in animal behavior*. Oxford University Press on Demand, 1996.

[10] Diego Darriba, Michael Weiß, and Alexandros Stamatakis. Prediction of missing sequences and branch lengths in phylogenomic data. *Bioinformatics*, 32(9):1331–1337, 2016.

[11] Francisco AP, Bugalho MF, Ramirez M, Carri{ç}o JA: Global optimal eBURST analysis of multilocus typing data using a graphic matroid approach. BMC Bioinf 2009., 10(152):.

[12] Alexandre P Francisco, Cátia Vaz, Pedro T Monteiro, José Melo-Cristino, Mário Ramirez, and João A Carriço. PHYLOViZ: phylogenetic inference and data visualization for sequence based typing methods. *BMC Bioinformatics*, 13(1):87, 2012.

[13] Marta Nascimento, Adriano Sousa, Mário Ramirez, Alexandre P Francisco, João A Carriço, and Cátia Vaz. PHYLOViZ 2.0: providing scalable data integration and visualization for multiple phylogenetic inference methods. *Bioinformatics (Oxford, England)*, page btw582, 2016.

[14] Naruya Saitou. *Introduction to evolutionary genomics*. Springer, 2013.

[15] Shahid H Bokhari and Daniel Janies. Reassortment networks for investigating the evolution of segmented viruses. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(2):288–298, 2010.

[16] Daniel Huson, Regula Rupp, and Celine Scornavacca. *Phylogenetic Networks: Concepts, Algorithms and Applications*, volume 1. 2011.

[17] R. W. Hamming. Error Detecting and Error Correcting Codes, 1950.

[18] J.A. Studier and K.J. Kepler. A note on the neighbour-joining method of Saitou and Nei. *Molecular Biology and Evolution*, 5(6):729–731, 1988.

[19] N Saitou and M Nei. The neighbour-joining method: a new method for reconstructing phylogenetic trees. *Mol Biol Evo*, 4(4):406–425, 1987.

[20] Fabio Pardi and Olivier Gascuel. Distance-based methods in phylogenetics. In Kliman R., editor, *Encyclopedia of Evolutionary Biology*, 1st Edition, pages 458–465. 2016.

[21] Travis J Wheeler. *Large-Scale Neighbor-Joining with NINJA*, pages 375–389. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[22] Thomas Mailund and Christian N. S. Pedersen. QuickJoin Fast Neighbour-Joining Tree Reconstruction. *Bioinformatics*, 20(17):3261–3262, 2004.

[23] Martin Simonsen, Thomas Mailund, and Christian N S Pedersen. *Rapid Neighbour-Joining*, pages 113–122. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[24] Martin Simonsen, Thomas Mailund, and C.N.S. N S Christian N S Pedersen. Building Very Large Neighbour-Joining Trees. *Proceedings of the First International Conference on Bioinformatics*, pages 26–32, 2010.

[25] J Wang, M Guo, and L L Xing. FastJoin, an improved neighbor-joining algorithm. *Genetics and Molecular Research*, 11(3):1909–1922, 2012.

[26] Kevin Howe, Alex Bateman, and Richard Durbin. QuickTree: building huge Neighbour-Joining trees of protein sequences. *Bioinformatics (Oxford, England)*, 18(11):1546–7, 2002.

[27] LR Foulds, MD Hendy, and David Penny. A graph theoretic approach to the development of minimal phylogenetic trees. *Journal of molecular Evolution*, 13(2):127–149, 1979.

[28] Alexandre P Francisco, Miguel Bugalho, Mário Ramirez, and João A Carriço. Global optimal eBURST analysis of multilocus typing data using a graphic matroid approach. *BMC Bioinformatics*, 10(1):152, 2009.

[29] E J Feil, B C Li, D M Aanensen, W P Hanage, and B G Spratt. eBURST: Inferring Patterns of Evolutionary Descent among Clusters of Related Bacterial Genotypes from Multilocus Sequence Typing Data. *Journal of Bacteriology*, 186, 2004.

[30] O Boruvka. On a minimal problem, 1926.

[31] Thomas H Cormen. *Introduction to algorithms*. MIT press, 2009.

[32] C H Papadimitriou and K Steiglitz. *Combinatorial Optimization. Dover*. 1998.

[33] H Whitney. On the abstract properties of linear dependence. *American Journal of Mathematics*, 57, 1935.

[34] W T Tutte. Lectures on matroids. *J Res Nat Bur Standards Sect B*, 69, 1965.

[35] J Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, 1, 1971.

[36] E J Feil, E C Holmes, D E Bessen, M S Chan, N P J Day, M C Enright, R Goldstein, D W Hood, A Kalia, and C E Moore. Recombination within natural populations of pathogenic bacteria: Short-term empirical estimates and long-term phylogenetic consequences. *Proceedings of the National Academy of Sciences*, 98, 2001.

[37] P Ruiz-Garbajosa, M J Bonten, D A Robinson, J Top, S R Nallapareddy, C Torres, T M Coque, R Canton, F Baquero, B E Murray, R del Campo, and R J Willems. Multilocus sequence typing scheme for Entrococcus faecalis reveals hospital-adapted genetic complexes in a background of high rates of recombination. *Journal of Clinical Microbiology*, 44, 2006.

[38] J B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7, 1956.

[39] R C Prim. Shortest connection networks and some generalizations. *Bell Syst Tech J*, 36, 1957.

[40] Andrew J Page, Ben Taylor, Aidan J Delaney, Jorge Soares, Torsten Seemann, Jacqueline A Keane, and Simon R Harris. SNP-sites: rapid efficient extraction of SNPs from multi-FASTA alignments. *Microbial Genomics*, 2(4), 2016.