

Fast phylogenetic inference from typing data

João André Carriço, Maxime Crochemore, Alexandre P. Francisco, Solon P. Pissis, Bruno Goncalves, Cátia Vaz

▶ **To cite this version:**

João André Carriço, Maxime Crochemore, Alexandre P. Francisco, Solon P. Pissis, Bruno Goncalves, et al.. Fast phylogenetic inference from typing data. Algorithms for Molecular Biology, 2018, 13, pp.4. 10.1186/s13015-017-0119-7 . hal-01785677

HAL Id: hal-01785677
https://hal.science/hal-01785677

Submitted on 4 May 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Algorithms for Molecular Biology

Fast phylogenetic inference from typing data

--Manuscript Draft--

Manuscript Number:	AMOB-D-17-00047R1
Full Title:	Fast phylogenetic inference from typing data
Article Type:	Research
Funding Information:	Fundação para a Ciência e a Tecnologia (TUBITAK/0004/2014) — Not applicable Royal Society International Exchanges Scheme — Not applicable Fundação para a Ciência e a Tecnologia (LISBOA-01-0145-FEDER-016394) — Not applicable Fundação para a Ciência e a Tecnologia (LISBOA-01-0145-FEDER-016417) — Not applicable Fundação para a Ciência e a Tecnologia (UID/CEC/500021/2013) — Not applicable European Food Safety Authority (GP/EFSA/AFSCO/2015/01/CT2) — Not applicable
Abstract:	Background: Microbial typing methods are commonly used to study the relatedness of bacterial strains. Sequence-based typing methods are a gold standard for epidemiological surveillance due to the inherent portability of sequence and allelic profile data, fast analysis times and their capacity to create common nomenclatures for strains or clones. This led to development of several novel methods and several databases being made available for many microbial species. With the mainstream use of High Throughput Sequencing, the amount of data being accumulated in these databases is huge, storing thousands of different profiles. On the other hand, computing genetic evolutionary distances among a set of typing profiles or taxa dominates the running time of many phylogenetic inference methods. It is important also to note that most of genetic evolution distance definitions rely, even if indirectly, on computing the pairwise Hamming distance among sequences or profiles. Results: We propose here an average-case linear-time algorithm to compute pairwise Hamming distances among a set of taxa under a given Hamming distance threshold. This article includes both a theoretical analysis and extensive experimental results concerning the proposed algorithm. We further show how this algorithm can be successfully integrated into a well known phylogenetic inference method, and how it can be used to speedup querying local phylogenetic patterns over large typing databases.
Corresponding Author:	Alexandre Francisco Universidade de Lisboa Instituto Superior Tecnico PORTUGAL
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	Universidade de Lisboa Instituto Superior Tecnico
Corresponding Author's Secondary Institution:	
First Author:	João Carriço
First Author Secondary Information:	
Order of Authors:	João Carriço Maxime Crochemore Alexandre Francisco

Powered by Editorial Manager® and ProduXion Manager® from Aries Systems Corporation

	Solon Pissis
	Bruno Ribeiro-Gonçalves
	Cátia Vaz, Ph.D
Order of Authors Secondary Information:	
Response to Reviewers:	Dear Editor, Dear Reviewers, Thank you very much for your comments on our manuscript. We tried our best to address all your comments and concerns. Please find below our detailed comments (starting with R:). Editor's comments (if any): Thank you for submitting a paper expanding upon your WABI submission. The reviewers felt that you satisfactorily > addressed the concerns raised in the reviews of the WABI paper and find value in the new additions you have made > here. They note only some minor remaining issues. Please see Reviewer #2's comments below for specific comments. I would ask that you revise the paper in accordance with the reviewer's suggested minor revisions and prepare a response to the critiques. I expect that we should be able to assess these revisions without the need to send the papers back to the reviewer again. R: We addressed suggested minor revisions, including missing references and typos within references. We revised also all the manuscript. The reviewers' critiques follow: Reviewer #1: The authors have satisfactorily addressed the comments that I raised for their original WABI submission and I am happy to recommend acceptance. R: Thank you. Reviewer #2: Overview: Distance-based phylogeny algorithms usually assume a matrix of pairwise distances between different taxa as input. However, there are algorithms that need only distances betweeen taxa that are sufficiently small, say smaller than some given k. The question is how to construct such a restricted similarity matrix as fast as possible. The submitted article tackles this question with Hamming distance as the measure between the genotype profile sequence representing each taxa. An O(md) average time optimal algorithm is given for small k, where d is the number of taxa and m is their length. The algorithm is a simple application of suffix arrays enhanced with LCP information and RMQ data structure. The main observation is that one can afford to output all length L>=m/k matching substring pairs between all profiles, and check which ones lead to real matches in O(k) time using constant time longest common extension queries implemented by RMQ on LCP array. The probability of false positives is small enough for small enough k, so that the running time is dominated by that of producing the output. Similar analyses have been conducted earlier for approximate string matching (Fredriksson and Navarro. Average-Optimal Multiple Approximate String Matching. CPM 2003). The resulting algorithm is plugged into an existing phylogeny tool that can exploit restricted matrices. Experiments show that the speed-up is significant and the simulations also confirm that the average case analysis assumptions are not too optimistic. A new application for the technique has been included to this extended journal version compared to the original conference paper. Namely, the application of querying typing databases is considered. In this application, a query pattern is searched in a database for approximate matches. The specialization of the all-pairs algorithm for this application is implemented and incorporated to INNUENDO Platform.

Powered by Editorial Manager® and ProduXion Manager® from Aries Systems Corporation

	Minor revision requests: Please add a reference to the approximate pattern matching literature, e.g. [Fredriksson and Navarro. Average-Optimal Multiple Approximate String Matching. CPM 2003], is one candidate, but some earlier work already contains similar analyses. With the added application of searching for query patterns this connection is even more evident. The analyses are really pretty much the same including the limitations on k when the approach works, so this connection should really be made visible, to give credit to the earlier work. R: We agree with the reviewer and we added the suggested reference, and another one, stating the relationship between the problems addressed in our work and approximate pattern matching. Both the statement and cited references may be found in Conclusions. Page 12, first line: Add "which is" or something like this to make the sentence complete. R: Thanks, we fixed the sentence adding "which is" as suggested. References: ??? in many places R: Thanks, some info was missing in our bib file. We revised all references and we believe that all are complete now. With my best regards, Alexandre Francisco

Powered by Editorial Manager® and ProduXion Manager® from Aries Systems Corporation

Manuscript Click here to download Manuscript paper.tex

Click here to view linked References

RESEARCH

Fast phylogenetic inference from typing data

João A Carriço[1], Maxime Crochemore[2], Alexandre P Francisco[3,4*], Solon P Pissis[2], Bruno Ribeiro-Gonçalves[1] and Cátia Vaz[3,5]

*Correspondence: aplf@ist.utl.pt
[3]INESC-ID Lisboa, Rua Alves Redol 9, 1000-029 Lisboa, PT
Full list of author information is available at the end of the article

Abstract

Background: Microbial typing methods are commonly used to study the relatedness of bacterial strains. Sequence-based typing methods are a gold standard for epidemiological surveillance due to the inherent portability of sequence and allelic profile data, fast analysis times and their capacity to create common nomenclatures for strains or clones. This led to development of several novel methods and several databases being made available for many microbial species. With the mainstream use of High Throughput Sequencing, the amount of data being accumulated in these databases is huge, storing thousands of different profiles. On the other hand, computing genetic evolutionary distances among a set of typing profiles or taxa dominates the running time of many phylogenetic inference methods. It is important also to note that most of genetic evolution distance definitions rely, even if indirectly, on computing the pairwise Hamming distance among sequences or profiles.

Results: We propose here an average-case linear-time algorithm to compute pairwise Hamming distances among a set of taxa under a given Hamming distance threshold. This article includes both a theoretical analysis and extensive experimental results concerning the proposed algorithm. We further show how this algorithm can be successfully integrated into a well known phylogenetic inference method, and how it can be used to speedup querying local phylogenetic patterns over large typing databases.

Keywords: computational biology; phylogenetic inference; Hamming distance

Background

Introduction

The evolutionary relationships between different species or *taxa* are usually inferred through known phylogenetic analysis techniques. Some of these techniques rely on

the inference of phylogenetic trees, which can be computed from DNA or Protein sequences, or from allelic profiles where the sequences of defined loci are abstracted to categorical indexes. The most popular method is MultiLocus Sequence Typing (MLST) [1] that typically uses seven 450 to 700 bp fragments of housekeeping genes for a given species. Phylogenetic trees are also used in other contexts, such as to understand the evolutionary history of gene families, to allow phylogenetic footprinting, to trace the origin and transmission of infectious diseases, or to study the co-evolution of hosts and parasites [2, 3].

In traditional phylogenetic methods, the process of phylogenetic inference starts with a multiple alignment of the sequences under study that is then corrected using models of DNA or Protein evolution. Tree-building methodologies can then be applied on the resulting distance matrix. These methods rely on some distance-based analysis of sequences or profiles [4].

Distance-based methods for phylogenetic analysis rely on a measure of genetic evolution distance, which is often defined directly or indirectly from the fraction of mismatches at aligned positions, with gaps either ignored or counted as mismatches. A first step of these methods is to compute this distance between all pairs of sequences. The simplest approach is to use the Hamming distance, also known as observed p-distance, defined as the number of positions at which two aligned sequences differ. Note that the Hamming distance between two sequences underestimates their true evolutionary distance and, thus, a correction formula based on some model of evolution is often used [2, 4]. Although distance-based methods not always produce the best tree for the data, usually they also incorporate an optimality criterion into the distance model for getting more plausible phylogenetic reconstructions, such as the minimum evolution criterion [5], the least squares criterion [6] or the clonal complexes expansion and diversification [7]. Nevertheless, this category of methods are much faster than Maximum Likelihood or Bayesian Inference Methods [8], making them excellent choices for the primary analysis of large data sets.

Most of the distance-based methods are agglomerative methods. They start with each sequence being a singleton cluster and, at each step, they join two clusters. The iterative process stops when all sequences are part of a single cluster, resulting in

a phylogenetic tree. At each step the candidate pair is selected taking into account the distance among clusters as well as the optimality criterion chosen to adjust it.

The computation of a distance matrix (2D array containing the pairwise distances between the elements of a set) is a common first step for distance-based methods, such as eBURST [9], goeBURST [10], Neighbor Joining [11] and UPGMA [12]. This particular step dominates the running time of most methods, taking $\Theta(md^2)$ time in general, d being the number of sequences or profiles and m the length of each sequence or profile. For large-scale datasets this running time may be quite problematic. And nowadays, with the mainstream use of High Throughput Sequencing, the amount of data being accumulated in typing databases is huge. It is common to find databases storing thousands of different profiles for a single microbial species, with each profile having thousands of loci [13, 14].

However, depending on application, on the underlying model of evolution and on the optimality criterion, it may not be strictly necessary to be aware of the complete distance matrix. There are methods that continue to provide optimal solutions without a complete matrix. For such methods, one may still consider a truncated distance matrix and several heuristics, combined with final local searches through topology rearrangements, to improve the running time [6]. The goeBURST algorithm, one of our use cases in this article, is an example of a method that can work with truncated distance matrices by construction, *i.e.*, one needs only to know which pairs are at Hamming distance at most k.

Our results

We propose here an average-case $\mathcal{O}(md)$-time and $\mathcal{O}(md)$-space algorithm to compute the pairs of sequences, among d sequences of length m, that are at distance at most k, when $k < \frac{(m-k-1)\cdot\log\sigma}{\log md}$, where σ is the size of the sequences alphabet. We support our result with both a theoretical analysis and an experimental evaluation on synthetic and real datasets of different data types (MLST, cgMLST, wgMLST and SNP). We further show that our method improves goeBURST, and that we can use it to speedup querying local phylogenetic patterns over large typing databases.

A preliminary version of this paper was presented at the Workshop on Algorithms in Bioinformatics (WABI) 2017 [15].

Methods

Closest pairs in linear time

Let P be the set of profiles (or sequences) each of length m, defined over an integer alphabet Σ, (*i.e.*, $\Sigma = \{1, \ldots, m^{O(1)}\}$), with $d = |P|$ and $\sigma = |\Sigma|$. Let also $H : P \times P \rightarrow \{0, \ldots, m\}$ be the function such that $H(u, v)$ is the Hamming distance between profiles $u, v \in P$. Given an integer threshold $0 < k < m$, the problem is to compute all pairs $u, v \in P$ such that $H(u, v) \leq k$, and the corresponding $H(u, v)$ value, faster than the $\Theta(md^2)$ time required to compute naïvely the complete distance matrix for the d profiles of length m.

We address this problem by indexing all profiles P using the suffix array (denoted by SA) and the longest common prefix (denoted by LCP) array [16]. We rely also on a range minimum queries (RMQ) data structure [17, 18] over the LCP array (denoted by RMQ_{LCP}). The problem is then solved in three main steps:

1. Index all profiles using the SA data structure.
2. Enumerate all candidate profile pairs given the maximum Hamming distance k.
3. Verify each candidate profile pair by checking if the associated Hamming distance is no more than k.

Table 1 summarizes the data structures and strategies followed in each step. Profiles are concatenated and indexed using SA. Depending on the strategy to be used, we further process the SA and build the LCP array and pre-process it for fast RMQ. This allows for enumerating candidate profile pairs and computing distances faster. In what follows, we detail the above steps and show how the data structures are used to improve the overall running time.

Step 1: Profile indexing

Profiles are concatenated and indexed in an SA in $\mathcal{O}(md)$ time and space [19, 20]. Let us denote this string by s. Since we only need to compute the distances between profiles that are at Hamming distance at most k, we can conceptually split each profile into k non-overlapping *blocks* of length $\mathcal{L} = \lfloor \frac{m}{k+1} \rfloor$ each. It is then folklore knowledge that if two profiles are within distance k, they must share at least one such block of length $\mathcal{L}$. Our approach is based on using the SA of s to efficiently

identify matching blocks among profile pairs. This lets us quickly filter in candidate profile pairs and filter out the ones that can never be part of the output.

Step 2: Candidate profile pairs enumeration

The candidate profile pairs enumeration step provides the pairs of profiles that do not differ in more than k positions, but it may include spurious pairs. Since SA is an ordered structure, a simple solution is to use a binary search approach. For each block of each profile, we can obtain in $\mathcal{O}(\mathcal{L} \log\ n)$ time, where $n = md$, all the suffixes that have that block as a prefix. If a given match is not aligned with the initial block, *i.e.* it does not occur at the same position in the respective profile, then it should be discarded. Otherwise, a candidate profile pair is reported. This searching procedure is done in $\mathcal{O}(dk\mathcal{L} \log n) = \mathcal{O}(n \log n)$ time.

Another solution relies on computing the LCP array: the longest common prefix between each pair of consecutive elements within the SA. This information can also be computed in $\mathcal{O}(n)$ time and space [21]. Since SA is an ordered structure, for the contiguous suffixes s_i, s_{i+1}, s_{i+2} of s, with $0 \leq i < n-2$, we have that the common prefix between s_i and s_{i+1} is at least as long as the common prefix of s_i and s_{i+2}. By construction, it is possible to get the position of each suffix in the corresponding profile in constant time. Then, we cluster the corresponding profiles of contiguous pairs if they have an LCP value greater than or equal to $\mathcal{L}$ *and* they are also aligned. This clustering procedure can be done in $\mathcal{O}(kd^2)$ time.

Step 3: Pairs verification

After getting the set of candidate profile pairs, a naïve solution would be to compute the distance for each pair of profiles by comparing them in linear time, *i.e.*, $\mathcal{O}(m)$ time. However, if we compute the LCP array of s, we can then perform a sequence of $\mathcal{O}(k)$ RMQ over the LCP array for checking if a pair of profiles is at distance at most k. These RMQ over the LCP array correspond to longest common prefix queries between a pair of suffixes of s. Since after a linear-time pre-processing over the LCP array, RMQ can be answered in constant time per query [17], we obtain a faster approach for computing the distances. This alternative approach takes $\mathcal{O}(k)$ time to verify each candidate profile pair instead of $\mathcal{O}(m)$ time.

Average-case analysis

Algorithm 1 below details the solution based on LCP clusters; and Theorem 1 shows that this algorithm runs in linear time on average using linear space. We rely here on well-known results concerning the linear-time construction of the SA [19, 20] and the LCP array [21], as well as the linear-time pre-processing for the RMQ data structure [18].

In what follows, LCP$[i]$, $i > 0$, stores the length of the longest common prefix of suffixes s_{i-1} and s_i of s, and $\text{RMQ}_{\text{LCP}}(i, j)$ returns the index of the smallest element in the subarray LCP$[i \ldots j]$ in constant time [18]. We rely also on some auxiliary subroutines; let $\mathcal{L} = \lfloor \frac{m}{k+1} \rfloor$:

Aligned(i) Let $\ell = i \mod m$, *i.e.*, the starting position of the suffix s_i within a profile. Then this subroutine returns $\ell/\mathcal{L}$ if ℓ is multiple of $\mathcal{L}$, and -1 otherwise.

HD(p_i, p_j, ℓ) Given two profiles p_i and p_j which share a substring of length $\mathcal{L}$, starting at index $\ell\mathcal{L}$, this subroutine computes the minimum of k and the Hamming distance between p_i and p_j. This subroutine relies on RMQ_{LCP} to find matches between p_i and p_j and, hence, it runs in $\mathcal{O}(k)$ time since it can terminate after k mismatches.

Theorem 1 *Given d profiles of length m each over an integer alphabet Σ of size $\sigma > 1$ with the letters of the profiles being independent and identically distributed random variables uniformly distributed over Σ, and the maximum Hamming distance $0 < k < m$, Algorithm 1 runs in $\mathcal{O}(md)$ average-case time and space if*

$$k < \frac{(m - k - 1) \cdot \log \sigma}{\log md}.$$

Proof Let us denote by s the string of length md obtained after concatenating the d profiles. The time and space required for constructing the SA and the LCP arrays for s and the RMQ data structure over the LCP array is $\mathcal{O}(md)$.

Let us denote by $\mathcal{B}$ the total number of blocks over s and by $\mathcal{L}$ the block length. We set $\mathcal{L} = \lfloor \frac{m}{k+1} \rfloor$ and thus we have that $\mathcal{B} = d\lfloor \frac{m}{\mathcal{L}} \rfloor$. Let us also denote by C a maximal set of indices over x satisfying the following:

1 the length of the longest common prefix between any two suffixes of s starting at these indices is at least $\mathcal{L}$;

Algorithm 1: Algorithm using LCP clusters.

Input: A set P of d profiles of length m each; an integer threshold $0 < k < m$.

Output: The set X of distinct pairs of profiles that are at Hamming distance at most k, *i.e.*, $X = \{(u, v) \in P \times P \mid u < v \text{ and } H(u, v) \leq k\}$.

Initialization: Let $s = s[0 \ldots n-1]$ be the string of length $n = md$ obtained after concatenating the d profiles, and $\mathcal{L} = \lfloor \frac{m}{k+1} \rfloor$. Construct the SA $\mathcal{S}$ for s, the LCP array for s and $\mathrm{RMQ}_{\mathrm{LCP}}$. Initialize a hash table HT to track verified pairs.

Candidate pairs enumeration:

$X := \emptyset$; $\ell_p := -1$; $C_t := \emptyset$, for $0 \leq t \leq k$

foreach $1 \leq i < n$ **do**

  $\ell := \mathrm{LCP}[i]$

  if $\ell \geq \mathcal{L}$ **then**

    $p_i := \lfloor \mathcal{S}[i]/m \rfloor$

    $x := \mathrm{Aligned}(i)$

    if $x \neq -1$ **then**

      $C_x := C_x \cup \{p_i\}$

    if $\ell_p = -1$ **then**

      $p_{i-1} := \lfloor \mathcal{S}[i-1]/m \rfloor$

      $x := \mathrm{Aligned}(i-1)$

      if $x \neq -1$ **then**

        $C_x := C_x \cup \{p_{i-1}\}$

    $\ell_p := \ell$

  else if $\ell_p \neq -1$ **then**

    Pairs enumeration:

    foreach C_t, *with* $0 \leq t \leq k$ **do**

      foreach $(p, q) \in C_t \times C_t : p < q$ **do**

        if $(p, q) \notin HT$ **then**

          $HT := HT \cup \{(p, q)\}$

          $\delta := \mathrm{HD}(p,\ q,\ t)$

          if $\delta \leq k$ **then**

            $X := X \cup \{(p, q)\}$

    $\ell_p := -1$; $C_t := \emptyset$, for $0 \leq t \leq k$

Finalize: Return the set X.

2 both of these suffixes start at the starting position of a block;

3 and both indices correspond to the starting position of the ith block in their profiles.

This can be done in $\mathcal{O}(md)$ time using the LCP array (lines 7-17). Processing all such sets C (lines 21-27) requires total time

$$\mathsf{PROC}_{i,j} \times \mathit{Pairs}$$

where $\mathsf{PROC}_{i,j}$ is the time required to process a pair i, j of elements of a set C, and *Pairs* is the sum of $|C|^2$ over all such sets C. We have that $\mathsf{PROC}_{i,j} = \mathcal{O}(k)$ by using RMQ over the LCP array. Additionally, by the stated assumption on the d profiles, the expected value for *Pairs* is no more than $\frac{\mathcal{B}d}{\sigma^{\mathcal{L}}}$: we have $\mathcal{B}$ blocks in total and each block can only match at most d other blocks by the conditions above.

Hence, the algorithm requires on average the following running time

$$\mathcal{O}(md + k \cdot \frac{\mathcal{B}d}{\sigma^{\mathcal{L}}}).$$

Let us analyze this further to obtain the relevant condition on k. We have the following:

$$k \cdot \frac{\mathcal{B}d}{\sigma^{\mathcal{L}}} = \frac{k \cdot \lfloor \frac{m}{\lfloor m/(k+1) \rfloor} \rfloor \cdot d^2}{\sigma^{\lfloor \frac{m}{k+1} \rfloor}} \leq \frac{k \cdot (\frac{m}{\lfloor m/(k+1) \rfloor}) \cdot d^2}{\sigma^{\frac{m}{k+1}-1}}.$$

Since $0 < k < m$ by hypothesis, we have the following:

$$\frac{k \cdot (\frac{m}{\lfloor m/(k+1) \rfloor}) \cdot d^2}{\sigma^{\frac{m}{k+1}-1}} \leq \frac{(md)^2}{\sigma^{\frac{m}{k+1}-1}}.$$

By some simple rearrangements we have that:

$$\frac{(md)^2}{\sigma^{\frac{m}{k+1}-1}} = \frac{(md)^2}{(md)^{\frac{\log \sigma}{\log md}(\frac{m}{k+1}-1)}} = (md)^{2-\frac{(m-k-1)\log \sigma}{(k+1)\log md}}.$$

Consequently, in the case when

$$k < \frac{(m-k-1) \cdot \log \sigma}{\log md}$$

the algorithm requires $\mathcal{O}(md)$ time on average. The extra space usage is clearly $\mathcal{O}(md)$. □

Use case 1: goeBURST algorithm

The distance matrix computation is a main step in distance-based methods for phylogenetic inference. This step dominates the running time of most methods, taking $\Theta(md^2)$ time, for d sequences of length m, since it must compute the distance among all sequence pairs. But for some methods, or when we are only interested in local phylogenies for sequences or profiles of interest, one does not need to know all pairwise distances for reconstructing a phylogenetic tree. The problem addressed

in this article was motivated by the goeBURST algorithm [10], our use case 1.

goeBURST is one of such methods for which one must know only the pairs of sequences that are at Hamming distance at most k. The solution proposed here can however be extended to other distance-based phylogenetic inference methods, that rely directly or indirectly on Hamming distance computations. Note that most methods either consider the Hamming distance or its correction accordingly to some formula based on some model of evolution [2, 4]. In both cases we must start by computing the Hamming distance among sequences, but not necessarily all of them [6].

The underlying model of goeBURST is as follows: a given genotype increases in frequency in the population as a consequence of a fitness advantage or of random genetic drift, becoming a founder clone in the population; and this increase is accompanied by a gradual diversification of that genotype, by mutation and recombination, forming a cluster of phylogenetic closely-related strains. This diversification of the "founding" genotype is reflected in the appearance of genetic profiles differing only in one housekeeping gene sequence from this genotype — single locus variants (SLVs). Further diversification of those SLVs will result in the appearance of variations of the original genotype with more than one difference in the allelic profile, *e.g.*, double and triple locus variants (DLVs and TLVs).

The problem solved by goeBURST can be stated as a graphic matroid optimization problem and, hence, it follows a classic greedy approach [22]. Given the maximum Hamming distance k, we can define a graph $G = (V, E)$, where $V = P$ (set of profiles) and $E = \{(u, v) \in V^2 \mid H(u, v) \leq k\}$. The main goal of goeBURST is then to compute a minimum spanning forest for G taking into account the distance H and a total order on links. It starts with a forest of singleton trees (each sequence/profile is a tree). Then it constructs the optimal forest by adding links connecting profiles in different trees in increasing order accordingly to the total order, similarly to what is done in the Kruskal's algorithm [23]. In the current implementation, a total order for links is implicitly defined based on the distance between sequences, on the number of SLVs, DLVs, TLVs, on the occurrence frequency of sequences, and on the assigned sequence identifier. With this total order, the construction of the tree consists of building a minimum spanning forest in a graph [23], where each sequence

is a node and the link weights are defined by the total order. By construction, the pairs at distance δ will be joined before the pairs at distance $\delta + 1$.

Use case 2: Querying typing databases

A related problem is querying typing databases for similar typing profiles. Given a set P of d profiles of length m each, a profile u not necessarily in P but with the same length m as those in P, and k such that $0 < k < m$, the problem is to find all profiles $v \in P$ such that $H(u, v) \leq k$. One may be also interested on local phylogenetic patterns, but those can be inferred from found profiles using for instance the goeBURST algorithm.

Once we define the value for k, we can address this problem as follows. We index all d profiles in the database as before in linear time $\mathcal{O}(md)$, and given a query profile u, we enumerate all candidate profiles v. We then verify as before all candidate pairs and we return only those satisfying $H(u, v) \leq k$.

Since u may not be in P, we rely neither on LCP clustering nor on RMQ. By using the SA we find candidate matches through binary search, identifying lower and higher bounds in the SA, as discussed before. Hence, given the $k + 1$ non-overlapping blocks of length $\mathcal{L} = \lfloor \frac{m}{k+1} \rfloor$ for u, we search for each one of them in $\mathcal{O}(\mathcal{L} \log md)$ time. Since we have $k+1$ blocks, it takes $\mathcal{O}(k\mathcal{L} \log md) = \mathcal{O}(m \log md)$ time to search for all $k + 1$ blocks in u. Finally, we can then verify and report all candidate profiles $v \in P$ as detailed in Algorithm 2.

Although, in the worst case, Algorithm 2 runs in time $\mathcal{O}(md + m \log md)$, as we may have d matches at most, we can prove a similar average case as in Theorem 1.

Theorem 2 *Given a profile u and a set of d profiles of length m each, all over an integer alphabet Σ of size $\sigma > 1$, with the letters of the profiles being independent and identically distributed random variables uniformly distributed over Σ, the SA for the string s of length md obtained after concatenating the d profiles, and the maximum Hamming distance $0 < k < m$, Algorithm 2 runs in $\mathcal{O}(m \log md)$ average-case time if*

$$k < \frac{(m - k - 1) \cdot \log \sigma + (k + 1) \cdot \log \log md}{\log md}.$$

Proof Let us denote by $\mathcal{B}$ the total number of blocks over s and by $\mathcal{L}$ the block length. We set $\mathcal{L} = \lfloor \frac{m}{k+1} \rfloor$ and thus we have that $\mathcal{B} = d \lfloor \frac{m}{\mathcal{L}} \rfloor$. By the stated assump-

Algorithm 2: Algorithm for querying typing databases.

Input: An SA $\mathcal{S}$ for a set P of d profiles of length m each, concatenated as a string s of length md; a profile u of length m; an integer threshold $0 < k < m$.

Output: The set X of distinct profiles that are at Hamming distance at most k from u, *i.e.*, $X = \{v \in P \mid H(u, v) \leq k\}$.

Initialization: Initialize a hash table HT to track verified profiles v and let $\mathcal{L} = \lfloor \frac{m}{k+1} \rfloor$.

1 $X := \emptyset$

2 **foreach** $0 \leq i \leq k$ **do**

3 $\quad \ell := \text{LowerBinSearch}(\mathcal{S}, s, u[i\mathcal{L}..(i+1)\mathcal{L} - 1])$

4 $\quad h := \text{HigherBinSearch}(\mathcal{S}, s, u[i\mathcal{L}..(i+1)\mathcal{L} - 1])$

5 $\quad$ **foreach** $\ell \leq j \leq h$ **do**

6 $\quad\quad$ **if** $\text{Aligned}(j) = i$ **then**

7 $\quad\quad\quad v := \lfloor \mathcal{S}[j]/m \rfloor$

8 $\quad\quad\quad$ **if** $v \notin HT$ **then**

9 $\quad\quad\quad\quad HT := HT \cup \{v\}$

10 $\quad\quad\quad\quad \delta := \text{H}(v, u)$

11 $\quad\quad\quad\quad$ **if** $\delta \leq k$ **then**

12 $\quad\quad\quad\quad\quad X := X \cup \{v\}$

13

14 **Finalize:** Return the set X.

tion on the profiles, the expected value for the number of profiles matching u is no more than $\frac{\mathcal{B}}{\sigma^{\mathcal{L}}}$: we have $\mathcal{B}$ blocks in total and each block can only match at most one other block in u (since *they must be aligned*; line 9). Moreover, since we are not relying on the LCP array in this case, the verification step (line 13) takes $\mathcal{O}(m)$ time. Hence, the algorithm requires on average the following running time

$$\mathcal{O}(m \log md + m \cdot \frac{\mathcal{B}}{\sigma^{\mathcal{L}}}).$$

Let us analyze this further to obtain the relevant condition on k. We have the following:

$$m \cdot \frac{\mathcal{B}}{\sigma^{\mathcal{L}}} = \frac{m \cdot \lfloor \frac{m}{\lfloor m/(k+1) \rfloor} \rfloor \cdot d}{\sigma^{\lfloor \frac{m}{k+1} \rfloor}} \leq \frac{m \cdot (\frac{m}{\lfloor m/(k+1) \rfloor}) \cdot d}{\sigma^{\frac{m}{k+1} - 1}} \leq \frac{m^2 d}{\sigma^{\frac{m}{k+1} - 1}}.$$

By some simple rearrangements we have that:

$$\frac{m^2 d}{\sigma^{\frac{m}{k+1} - 1}} = \frac{m^2 d}{(md)^{\frac{\log \sigma}{\log md}(\frac{m}{k+1} - 1)}} = m(md)^{1 - \frac{(m-k-1)\log \sigma}{(k+1)\log md}}.$$

Consequently, in the case when

$$k < \frac{(m - k - 1) \cdot \log \sigma + (k + 1) \cdot \log \log md}{\log md}$$

the algorithm requires $\mathcal{O}(m \log md)$ time on average. □

This algorithm was implemented and integrated in INNUENDO Platform, which is publicly available [24]. The INNUENDO Platform is an infrastructure that provides the required framework for data analyses from bacterial raw reads sequencing data quality insurance to the integration of epidemiological data and visualization. As such, rapid methods for classification and search for closely related strains are a necessity for quick navigation through the platform database entries. More information about the project can be found at its website [25].

As a starting point and for the purpose of this study, a subset of 2312 wgMLST profiles of *Escherichia coli* retrieved from Enterobase [13] were included in the INNUENDO database as well as their ancillary data and predefined core-genome cluster classification. Two tab-separated files containing the wgMLST and cgMLST profiles for the *Escherichia coli* strains were also created to allow storing information on the currently available profiles and for updating with profiles that will become available upon the platform analyses.

One of two index files are used depending on the type of search we want to perform: classification or search for k-closest. The cgMLST index file is used for strain classification, which relies on a nomenclature designed for the cgMLST profiles. As such, and since a pre-classification was performed on the database of *Escherichia coli* strains, we continued using it for comparison purposes. However, when searching for the k-closest profiles, we take into consideration all targets available in the wgMLST profiles using the wgMLST index file for a higher discriminatory power.

Each time a new profile is generated from the platform, it requires classification. The INNUENDO Platform performs the classification step based on the approach described in our use case 2 with a given maximum of k differences over core genes. It uses the cgMLST index file for the search since the classification is constructed based on those number of loci. If the method returns at least one match, it classifies the new profile with the classification of the closest. If not, a new classification is assigned. A new entry is then added to the INNUENDO database as well as to the cgMLST and wgMLST profiles files and the index files are updated.

In the case of the search for the k-closest, it is useful to define the input data for visualization methods according to a defined number of differences on close strains.

For each profile used as input for the search, the method searches for the k-closest strains considering at most k differences among all wgMLST loci. Since duplicate matches can occur between the profiles used for each search, the final file used as input for the visualization methods is the intersection of the results of the k-closest profiles between each input strain. The set of strain identifiers are then used to query the INNUENDO database to get the profiles and ancillary data to be sent to PHYLOViZ Online [26] for further analysis, namely with the goeBURST algorithm.

The drawback of using this method for classification and search is the need for rebuilding the index each time there is a new profile, which will depend on the number of profile entries on the database. Nevertheless, the number of updates is rather smaller compared to the number of queries and the index can be build in the background, with search functionalities still using the old index during the process. In our implementation, the index and related data structures are serialized in secondary memory and they are accessed by mapping them into memory. The implementation of the underlying tool is made publicly available [27].

The above described approaches in combination with the features offered by the INNUENDO Platform allow microbiologists to quickly and efficiently search for strains close to their strain of interest, allowing a more targeted, focused and simple visualization of results.

Experimental evaluation

We evaluated the proposed approach to compute the pairs of profiles at distance at most k using both real and synthetic datasets. We used real datasets obtained through different typing schemas, namely whole-genome multi-locus sequence typing (wgMLST) data, core-genome multi-locus sequence typing (cgMLST) data, and single-nucleotide polymorphism (SNP) data. Table 2 summarizes the real datasets used. We should note that wgMLST and cgMLST datasets contain sequences of integers, where each column corresponds to a locus and different values in the same column denote different alleles. Synthetic datasets comprise sets of binary sequences of variable length, uniformly sampled, allowing us to validate our theoretical findings.

We implemented both versions described above in the C programming language: one based on binary search over the SA; and another one based on finding clusters

in the LCP array. Since allelic profiles can be either string of letters or sequences of integers, we relied on `libdivsufsort` library [28] and `qsufsort` code [29, 30], respectively. For RMQ over the LCP array, we implemented a fast well-known solution that uses constant time per query and linearithmic space for pre-processing [17].

All tests were conducted on a machine running Linux, with an Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz (8 cores, cache 32KB/4096KB) and with 32GB of RAM. All binaries where produced using GCC 5.3 with full optimization enabled.

Synthetic datasets

We first present results with synthetic data for different values of d, m and k. All synthetic sequences are binary sequences uniformly sampled. Results presented in this section were averaged over ten runs and for five different sets of synthetic data.

The bound proved in Theorem 1 was verified in practice. For k satisfying the conditions in Theorem 1, the running time of our implementation grows almost linearly with n, the size of the input. We can observe in Fig. 1 a growth slightly above linear. Since we included the time for constructing the SA, the LCP array and the RMQ data structure, with the last one in linearithmic time, that was expected.

We also tested our method for values of k exceeding the bound shown in Theorem 1. For $d = m = 4096$ and a binary alphabet, the bound for k given in Theorem 1 is no more than $\lfloor m/(2 \log m) \rfloor = 170$. For k above this bound we expect that proposed approaches are no longer competitive with the naïve approach. As shown in Fig. 2, for $k > 250$ and $k > 270$ respectively, both limits above the predicted bound, the running time for both computing pairwise distances by finding lower and higher bounds in the SA, and by processing LCP based clusters, becomes slower than the running time of the naïve approach.

In Fig. 3 we have the running time as a function of the number d of profiles, for different values of m and for k satisfying the bound given in Theorem 1. The running time for the naïve approach grows quadratically with d, while it grows linearly for both computing pairwise distances by finding lower and higher bounds in the SA, and by processing LCP based clusters. Hence, for synthetic data, as described by Theorem 1, the result holds.

Real datasets

For each dataset in Table 2, we ranged the threshold k accordingly and compared the approaches discussed in methods section with the naïve approach that computes the distance for all sequence pairs. Results are provided in Table 3.

In most cases, the approach based on the LCP clusters is the fastest up to two orders of magnitude compared to the naïve approach. As expected, in the case when data are not uniformly random, our method works reasonably well for smaller values of k than the ones implied by the bound in Theorem 1. As an example, the upper bound on k for *C. jejuni* would be around 200, but the running time for the naïve approach is already better for $k = 64$. We should note however that the number of candidate profile pairs at Hamming distance at most k is much higher than the expected number when data are uniformly random. This tells us that we can design a simple hybrid scheme that chooses a strategy (naïve or the proposed method) depending on the nature of the input data. It seems also to point out clustering effects on profile dissimilarities, which we may exploit to improve our results. We leave both tasks as future work for the full version of this article.

We incorporated the approach based on finding lower and higher bounds in the SA in the implementation of goeBURST algorithm, discussed in methods section. We did not incorporate the approach based on the LCP clusters as the running time did not improve much as observed above. Since running times are similar to those reported in Table 3, we discuss only the running time for *C. jejuni*. We need only to index the input once. We can then use the index in the different stages of the algorithm and for different values of k. In the particular case of goeBURST, we use the index twice: once for computing the number of neighbors at a given distance, used for untying links according to the total order discussed in the description of goeBURST algorithm in methods section, and a second time for enumerating pairs at distance below a given threshold. Note that the goeBURST algorithm does not aim to link all nodes, but to identify clonal complexes (or connected components) for a given threshold on the distance among profiles [10]. In the case of *C. jejuni* dataset, and for $k = 52$, the running time is around 36 seconds, while the naïve approach takes around 115 seconds, yielding a three-fold speedup. In this case we get several connected components, *i.e.*, several trees, connecting the most similar profiles. We provide the tree for the largest component in Fig. 4, where each node

represents a profile. The nodes are colored according to one of the loci for which profiles in this cluster differ. Note that this tree is optimal with respect to the criterion used by the goeBURST algorithm, not being affected by the threshold on the distance. In fact, since this problem is a graphic matroid, the trees found for a given threshold will be always subtrees of the trees found for larger thresholds [22]. Comparing this tree with other inference methods is beyond the scope of this article; the focus here was on the faster computation of an optimal tree under this model.

In many studies, the computation of trees based on pairwise distances below a given threshold, usually small compared with the total number of loci, combined with ancillary data, such as antibiotic resistance and host information, allows microbiologists to uncover evolution patterns and study the mechanisms underlying the transmission of infectious diseases [31].

Conclusions

Most distance-based phylogenetic inference methods rely directly or indirectly on Hamming distance computations. The computation of a distance matrix is a common first step for such methods, taking $\Theta(md^2)$ time in general, with d being the number of sequences or profiles and m the length of each sequence or profile. For large-scale datasets this running time may be problematic; however, for some methods, we can avoid to compute all-pairs distances [6].

We addressed this problem when only a truncated distance matrix is needed, *i.e.*, one needs to know only which pairs are at Hamming distance at most k. This problem was motivated by the goeBURST algorithm [10], which relies on a truncated distance matrix by construction. Both the problem and techniques discussed here are related to average-case approximate string matching [32, 33]. We proposed here an average-case linear-time and linear-space algorithm to compute the pairs of sequences or profiles that are at Hamming distance at most k, when $k < \frac{(m-k-1)\cdot\log\sigma}{\log md}$, where σ is the size of the alphabet. We integrated our solution in goeBURST demonstrating its effectiveness using both real and synthetic datasets.

We must note however that our analysis holds for uniformly random sequences and, hence, as observed with real data, the presented bound may be optimistic. It is thus interesting to investigate how to address this problem taking into account local conserved regions within sequences. Moreover, it might be interesting to consider

in the analysis null models such as those used to evaluate the accuracy of distance-based phylogenetic inference methods [4].

The proposed approach is particularly useful when one is interested in local phylogenies, *i.e.*, local patterns of evolution, such as searching for similar sequences or profiles in large typing databases, as in our use case 2. In this case we do not need to construct full phylogenetic trees, with tens of thousands of taxa. We can focus our search on the most similar sequences or profiles, within a given threshold k. There are however some issues to be solved in this scenario, namely, dynamic updating of the data structures used in our algorithm. Note that after querying a database, if new sequences or profiles are identified, then we should be able to add them while keeping our data structures updated. Although more complex and dynamic data structures are known, a technique proposed recently for adding dynamism to otherwise static data structures can be useful to address this issue [34]. This and other challenges raised above are left as future work.

Competing interests

The authors declare that they have no competing interests.

Author's contributions

MC, APF, SPP and CV conceived the study and contributed for the design and analysis of the methods and experimental evaluation. APF, SPP and CV implemented Algorithm 1 and run the experiments. JAC conceived the case study 2 and contributed with the biological background. APF and BRG implemented Algorithm 2 and integrated it in INNUENDO Platform. All authors contributed to the writing of the manuscript. All authors have read and approved the final manuscript.

Acknowledgements

This work was partly supported by the Royal Society International Exchanges Scheme, and by the following projects: BacGenTrack (TUBITAK/0004/2014) funded by FCT (Fundação para a Ciência e a Tecnologia) / Scientific and Technological Research Council of Turkey (Türkiye Bilimsel ve Teknolojik Araşrrma Kurumu, TÜBİTAK), PRECISE (LISBOA-01-0145-FEDER-016394) and ONEIDA (LISBOA-01-0145-FEDER-016417) projects co-funded by FEEI (Fundos Europeus Estruturais e de Investimento) from "Programa Operacional Regional Lisboa 2020" and by national funds from FCT, UID/CEC/500021/2013 funded by national funds from FCT, and INNUENDO project [25] co-funded by the European Food Safety Authority (EFSA), grant agreement GP/EFSA/AFSCO/2015/01/CT2 ("New approaches in identifying and characterizing microbial and chemical hazards"). The conclusions, findings, and opinions expressed in this review paper reflect only the view of the authors and not the official position of the European Food Safety Authority (EFSA).

Author details

[1]Faculdade de Medicina, Instituto de Microbiologia and Instituto de Medicina Molecular, Universidade de Lisboa, Lisboa, PT. [2]Department of Informatics, King's College London, London, UK. [3]INESC-ID Lisboa, Rua Alves Redol 9, 1000-029 Lisboa, PT. [4]Instituto Superior Técnico, Universidade de Lisboa, Lisboa, PT. [5]Instituto Superior de Engenharia de Lisboa, Instituto Politécnico de Lisboa, Lisboa, PT.

References

1. Maiden, M.C., Bygraves, J.A., Feil, E.J., Morelli, G., Russell, J.E., Urwin, R., Zhang, Q., Zhou, J., Zurth, K., Caugant, D.A., Feavers, I.M., Achtman, M., Spratt, B.G.: Multilocus sequence typing: a portable approach to the identification of clones within populations of pathogenic microorganisms. Proceedings of the National Academy of Sciences of the United States of America **95**(6), 3140–3145 (1998)
2. Huson, D.H., Rupp, R., Scornavacca, C.: Phylogenetic Networks: Concepts, Algorithms and Applications. Cambridge University Press, New York, NY, USA (2010). doi:10.1017/CBO9780511974076
3. Robinson, D.A., Feil, E.J., Falush, D.: Bacterial Population Genetics in Infectious Disease. John Wiley & Sons, Hoboken, NJ, USA (2010). doi:10.1002/9780470600122
4. Saitou, N.: Introduction to Evolutionary Genomics. Springer, London (2013). doi:10.1007/978-1-4471-5304-7
5. Desper, R., Gascuel, O.: Fast and accurate phylogeny reconstruction algorithms based on the minimum-evolution principle. Journal of Computational Biology **9**(5), 687–705 (2002). doi:10.1089/106652702761034136
6. Pardi, F., Gascuel, O.: Distance-based methods in phylogenetics. In: Encyclopedia of Evolutionary Biology, pp. 458–465. Elsevier, Oxford, MA, USA (2016). doi:10.1016/B978-0-12-800049-6.00206-7
7. Feil, E.J., Holmes, E.C., Bessen, D.E., Chan, M.-S., Day, N.P., Enright, M.C., Goldstein, R., Hood, D.W., Kalia, A., Moore, C.E., *et al.*: Recombination within natural populations of pathogenic bacteria: short-term empirical estimates and long-term phylogenetic consequences. Proceedings of the National Academy of Sciences **98**(1), 182–187 (2001). doi:10.1073/pnas.98.1.182
8. Yang, Z., Rannala, B.: Molecular phylogenetics: principles and practice. Nature Reviews Genetics **13**(5), 303–314 (2012)
9. Feil, E.J., Li, B.C., Aanensen, D.M., Hanage, W.P., Spratt, B.G.: eBURST: inferring patterns of evolutionary descent among clusters of related bacterial genotypes from multilocus sequence typing data. Journal of Bacteriology **186**(5), 1518–1530 (2004). doi:10.1128/JB.186.5.1518-1530.2004
10. Francisco, A.P., Bugalho, M., Ramirez, M., Carriço, J.: Global optimal eBURST analysis of multilocus typing data using a graphic matroid approach. BMC Bioinformatics **10**(1) (2009). doi:10.1186/1471-2105-10-152
11. Saitou, N., Nei, M.: The neighbor-joining method: a new method for reconstructing phylogenetic trees. Molecular Biology and Evolution **4**(4), 406–425 (1987). doi:10.1093/oxfordjournals.molbev.a040454
12. Sokal, R.R.: A statistical method for evaluating systematic relationships. Univ Kans Sci Bull **38**, 1409–1438 (1958)
13. Sergean, M., Zhou, Z., Alikhan, N.-F., Achtman, M.: EnteroBase. Accessed on 31 October 2017. https://enterobase.warwick.ac.uk/
14. Jolley, K.A., Maiden, M.C.J.: BIGSdb: Scalable analysis of bacterial genome variation at the population level. BMC Bioinformatics **11**, 595 (2010)
15. Crochemore, M., Francisco, A.P., Pissis, S.P., Vaz, C.: Towards Distance-Based Phylogenetic Inference in Average-Case Linear-Time. In: Schwartz, R., Reinert, K. (eds.) 17th International Workshop on Algorithms in Bioinformatics (WABI 2017). Leibniz International Proceedings in Informatics (LIPIcs), vol. 88, pp. 9–1914. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2017). doi:10.4230/LIPIcs.WABI.2017.9. http://drops.dagstuhl.de/opus/volltexte/2017/7652
16. Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. SIAM Journal on Computing **22**(5), 935–948 (1993). doi:10.1137/0222058
17. Bender, M.A., Farach-Colton, M.: The LCA problem revisited. In: LATIN 2000: Theoretical Informatics: 4th Latin American Symposium. Lecture Notes in Computer Science, vol. 1776, pp. 88–94. Springer, Berlin, Heidelberg (2000). doi:10.1007/10719839_9
18. Bender, M.A., Farach-Colton, M., Pemmasani, G., Skiena, S., Sumazin, P.: Lowest common ancestors in trees and directed acyclic graphs. Journal of Algorithms **57**(2), 75–94 (2005). doi:10.1016/j.jalgor.2005.08.001
19. Kärkkäinen, J., Sanders, P., Burkhardt, S.: Linear work suffix array construction. Journal of ACM **53**(6), 918–936 (2006). doi:10.1145/1217856.1217858
20. Ko, P., Aluru, S.: Space efficient linear time construction of suffix arrays. In: Annual Symposium on Combinatorial Pattern Matching. Lecture Notes in Computer Science, vol. 2676, pp. 200–210. Springer, Berlin, Heidelberg (2003). doi:10.1016/j.jda.2004.08.002

21. Kasai, T., Lee, G., Arimura, H., Arikawa, S., Park, K.: Linear-time longest-common-prefix computation in suffix arrays and its applications. In: Annual Symposium on Combinatorial Pattern Matching, pp. 181–192 (2001). doi:10.1007/3-540-48194-X. Springer
22. Papadimitriou, C.H., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1982)
23. Kruskal, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical Society **7**(1), 48–50 (1956). doi:10.2307/2033241
24. B-UMMI: INNUENDO Platform. Accessed on 31 October 2017. https://github.com/B-UMMI/INNUENDO
25. INNUENDO: A novel cross-sectorial platform for the integration of genomics in surveillance of foodborne pathogens. Accessed on 31 October 2017. http://www.innuendoweb.org/
26. Ribeiro-Gonçalves, B., Francisco, A.P., Vaz, C., Ramirez, M., Carriço, J.A.: Phyloviz online: web-based tool for visualization, phylogenetic inference, analysis and sharing of minimum spanning trees. Nucleic Acids Research **44**(Webserver-Issue), 246–251 (2016). doi:10.1093/nar/gkw359
27. B-UMMI: Fast MLST searching and querying. Accessed on 31 October 2017. https://github.com/B-UMMI/fast-mlst
28. Mori, Y.: A lightweight suffix-sorting library. Accessed on 31 October 2017. https://github.com/y-256/libdivsufsort
29. Larsson, N.J., Sadakane, K.: Suffix sorting implementation to accompany the paper *Faster Suffix Sorting*. Accessed on 31 October 2017. http://www.larsson.dogma.net/qsufsort.c
30. Larsson, N.J., Sadakane, K.: Faster suffix sorting. Theor. Comput. Sci. **387**(3), 258–272 (2007). doi:10.1016/j.tcs.2007.07.017
31. Francisco, A.P., Vaz, C., Monteiro, P.T., Melo-Cristino, J., Ramirez, M., Carriço, J.A.: PHYLOViZ: phylogenetic inference and data visualization for sequence based typing methods. BMC Bioinformatics **13**(1), 87 (2012). doi:10.1186/1471-2105-13-87
32. Fredriksson, K., Navarro, G.: Average-optimal single and multiple approximate string matching. ACM Journal of Experimental Algorithmics **9** (2004). doi:10.1145/1005813.1041513
33. Barton, C., Iliopoulos, C.S., Pissis, S.P.: Fast algorithms for approximate circular string matching. Algorithms for Molecular Biology **9**, 9 (2014). doi:10.1186/1748-7188-9-9
34. Munro, J.I., Nekrich, Y., Vitter, J.S.: Dynamic data structures for document collections and graphs. In: Proceedings of the 34th ACM Symposium on Principles of Database Systems, pp. 277–289. ACM, New York, NY, USA (2015). doi:10.1145/2745754.2745778
35. Nascimento, M., Sousa, A., Ramirez, M., Francisco, A.P., Carriço, J.A., Vaz, C.: PHYLOViZ 2.0: providing scalable data integration and visualization for multiple phylogenetic inference methods. Bioinformatics **33**(1), 128–129 (2017). doi:10.1093/bioinformatics/btw582
36. Page, A.J., Taylor, B., Delaney, A.J., Soares, J., Seemann, T., Keane, J.A., Harris, S.R.: SNP-sites: rapid efficient extraction of SNPs from multi-FASTA alignments. Microbial Genomics **2**(4) (2016). doi:10.1099/mgen.0.000056
37. Croucher, N.J., Finkelstein, J.A., Pelton, S.I., Mitchell, P.K., Lee, G.M., Parkhill, J., Bentley, S.D., Hanage, W.P., Lipsitch, M.: Population genomics of post-vaccine changes in pneumococcal epidemiology. Nature Genetics **45**(6), 656–663 (2013). doi:10.1038/ng.2625
38. Chewapreecha, C., Harris, S.R., Croucher, N.J., Turner, C., Marttinen, P., Cheng, L., Pessia, A., Aanensen, D.M., Mather, A.E., Page, A.J., Salter, S.J., Harris, D., Nosten, F., Goldblatt, D., Corander, J., Parkhill, J., Turner, P., Bentley, S.D.: Dense genomic sampling identifies highways of pneumococcal recombination. Nature Genetics **46**(3), 305–309 (2014). doi:10.1038/ng.2895
39. National Center for Biotechnology Information: GeneBank. Accessed on 31 October 2017. ftp://ftp.ncbi.nih.gov/genomes/archive/old_genbank/Bacteria/

Figures

Figure 1 Synthetic datasets, with $\sigma = 2$ and $k = \lfloor m/(2 \log m) \rfloor$ according to Theorem 1. Running time for computing pairwise distances by finding lower and higher bounds in the SA, and by processing LCP based clusters, as function of the input size $n = dm$.

Figure 2 Synthetic datasets, with $\sigma = 2$ and $m = 4096$. Running time for computing pairwise distances by finding lower and higher bounds in the SA, and by processing LCP based clusters, as function of the number d of profiles and for different values of k.

Figure 3 Synthetic datasets, with $\sigma = 2$ and $k = \lfloor m/(2 \log m) \rfloor$ according to Theorem 1. Running time for computing pairwise distances naïvely, by finding lower and higher bounds in the SA, and by processing LCP based clusters, as a function of the number d of profiles.

Figure 4 The tree inferred for the largest connected component found with $k = 52$ for the *C. jejuni* dataset. Image produced by PHYLOViZ [35].

Tables

Table 1 Data structures used in our approach for each step.

Profile indexing	Candidate profile pairs enumeration	Pairs verification
Suffix array	Binary search	Naïve
	LCP based clusters	$\mathrm{RMQ}_{\mathrm{LCP}}$

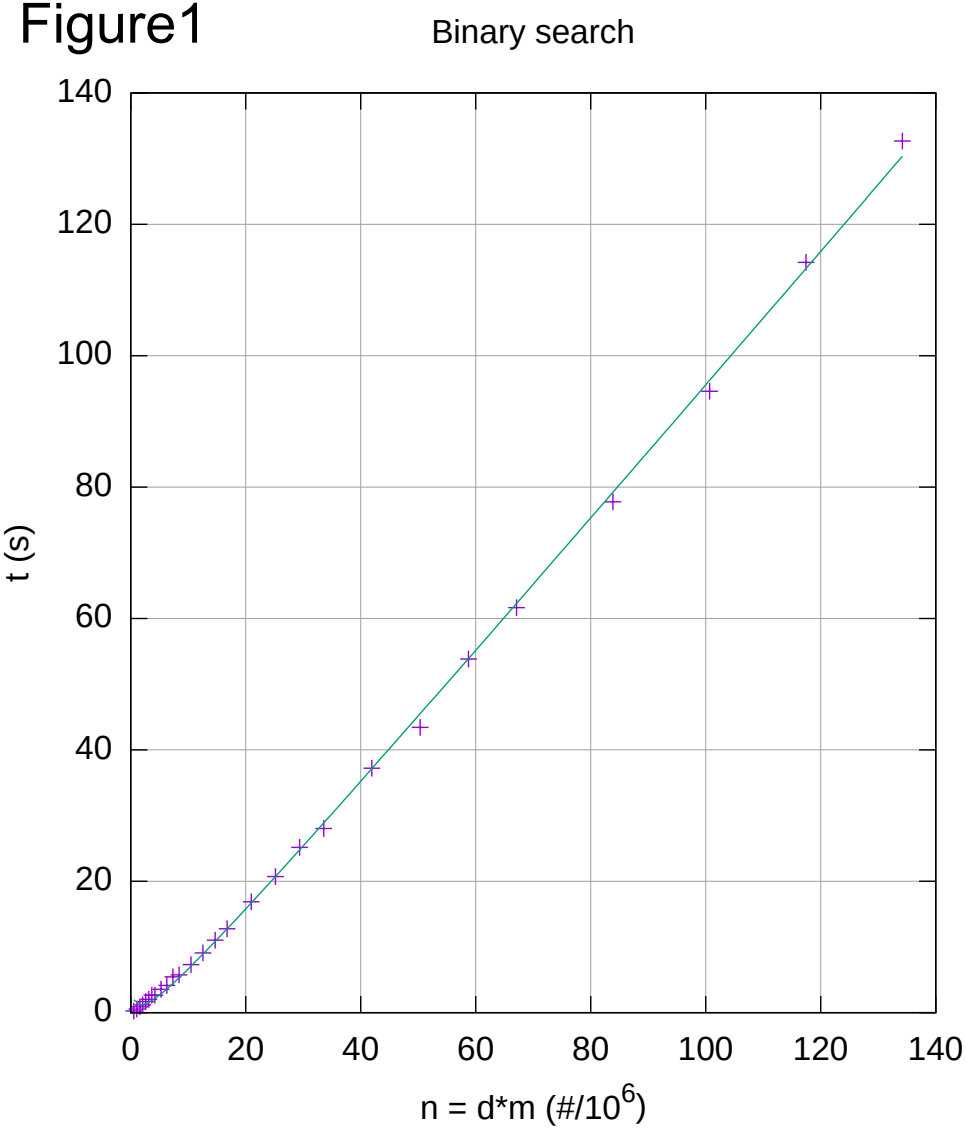
Table 2 Real datasets used in the experimental evaluation. (*)Dataset provided by the Molecular Microbiology and Infection Unit, IMM.

Dataset	Typing method	Profile length	Number of distinct elements	Reference
Campylobacter jejuni	wgMLST	5446	5669	(*)
Salmonella enterica	wgMLST	3002	6861	[13]
Salmonella typhi	SNP	22143	1534	[36]
Streptococcus pneumoniae	cgMLST	235	1968	[37, 38, 39]

Table 3 Time and percentage of pairs processed for each method and dataset.

Dataset	k	Naïve		Binary search		LCP clusters	
		t (s)	pairs (%)	t (s)	pairs (%)	t (s)	pairs (%)
C. jejuni	8	108.59	100	0.22	0.06	**0.17**	0.06
	16	109.30	100	0.48	0.32	**0.34**	0.32
	32	108.60	100	3.52	5.45	**2.67**	5.45
	64	**108.60**	100	231.05	99.98	162.36	99.98
S. enterica	8	89.85	100	1.04	2.37	**0.95**	2.37
	16	87.26	100	7.16	12.69	**6.73**	12.69
	32	85.36	100	36.29	33.22	**30.76**	33.22
	64	**84.63**	100	254.45	82.44	187.15	82.44
S. typhi	89	28.83	100	16.63	91.48	**12.02**	91.48
	178	**28.32**	100	46.98	99.91	32.03	99.91
	890	**30.04**	100	113.57	100	129.14	100
S. pneumoniae	8	0.56	100	0.02	0.93	**0.02**	0.93
	16	0.57	100	0.05	1.71	**0.04**	1.71
	32	0.56	100	0.20	4.42	**0.15**	4.42
	64	**0.58**	100	5.63	73.36	5.01	73.36

Figure1



Click here to download Figure rtest.pdf

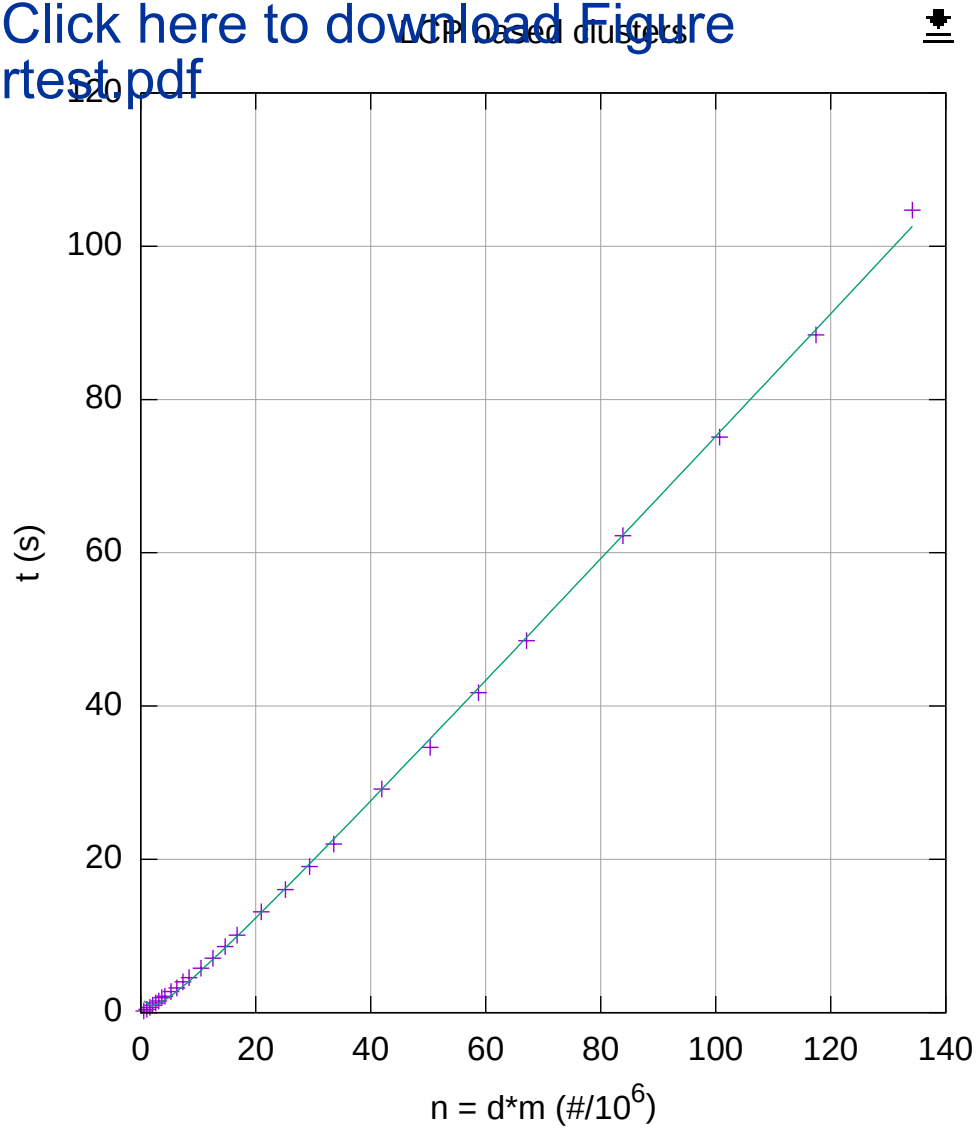
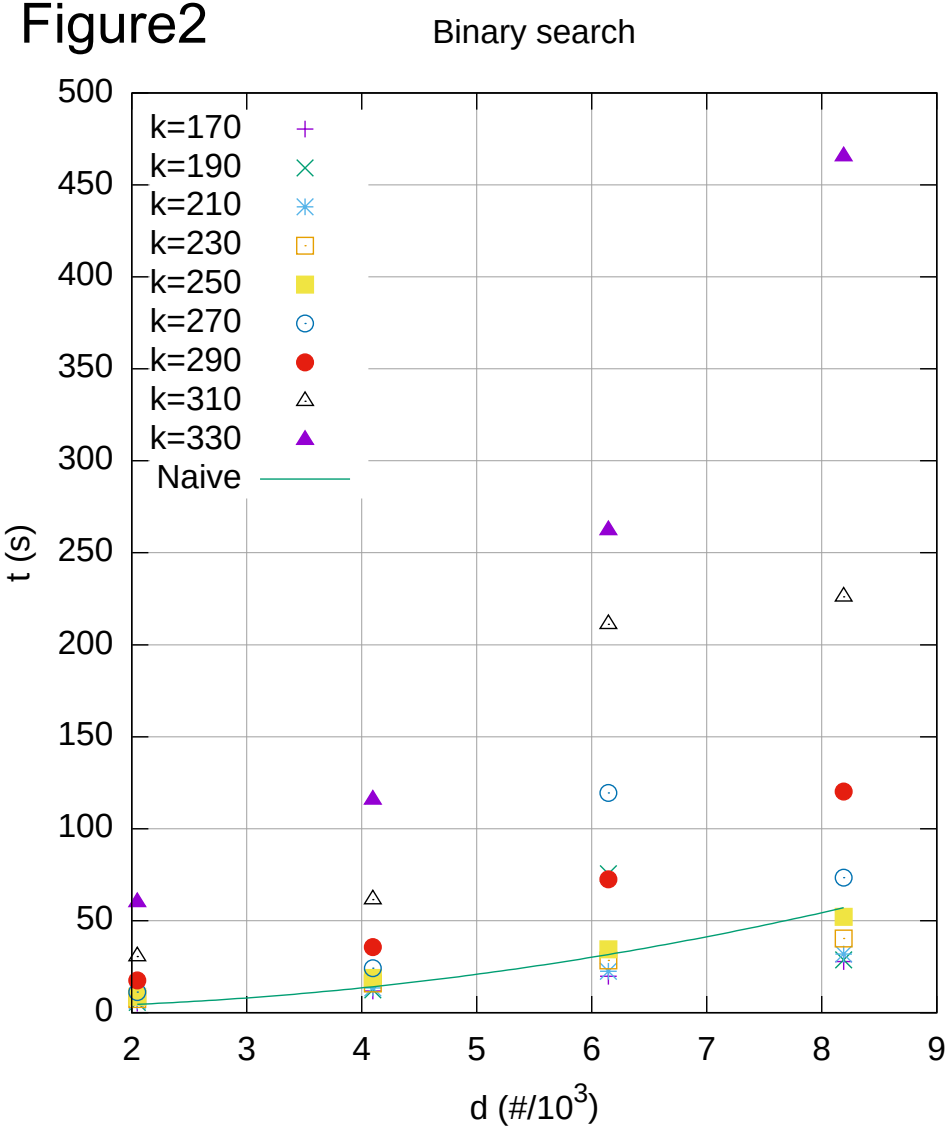


Figure2



Click here to download Figure rtest_k.pdf

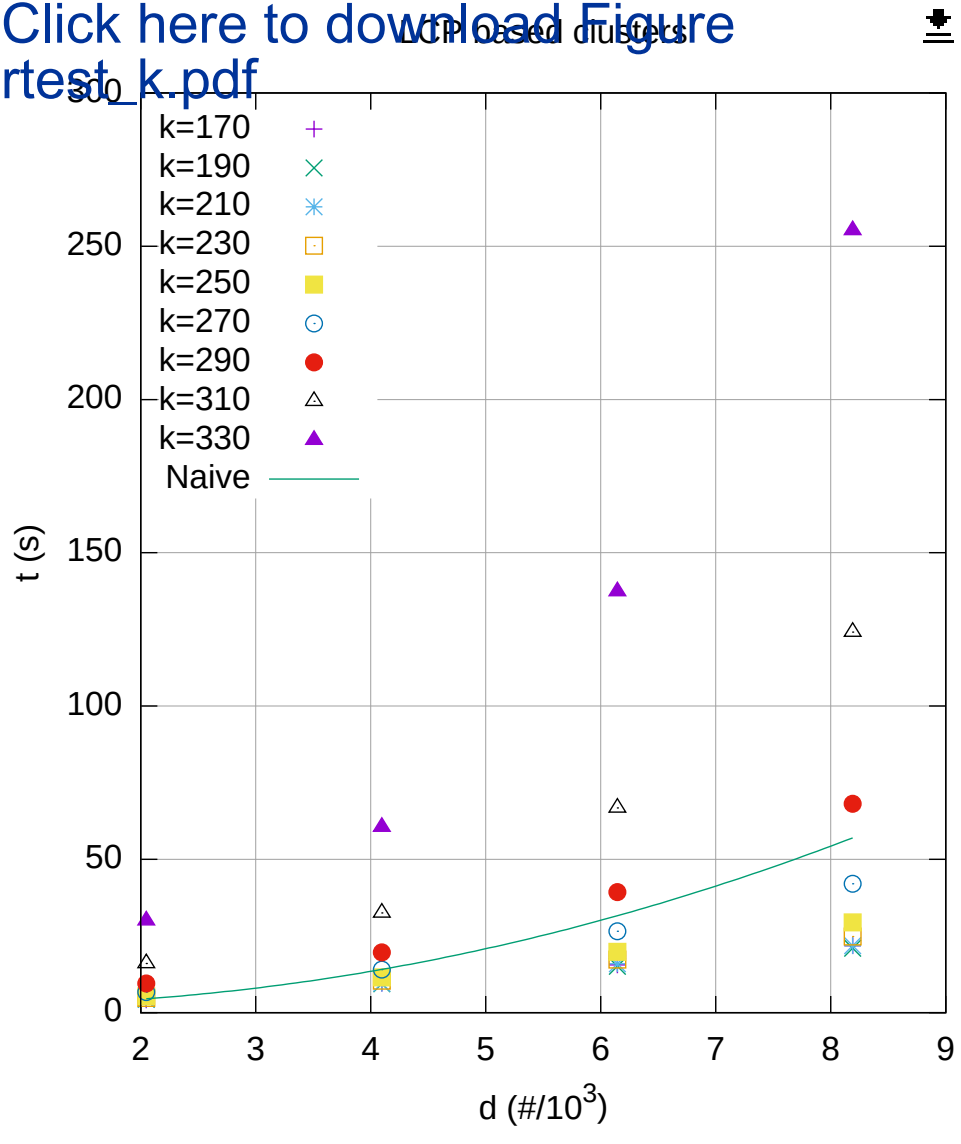
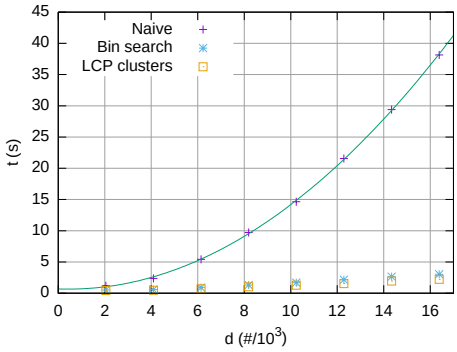
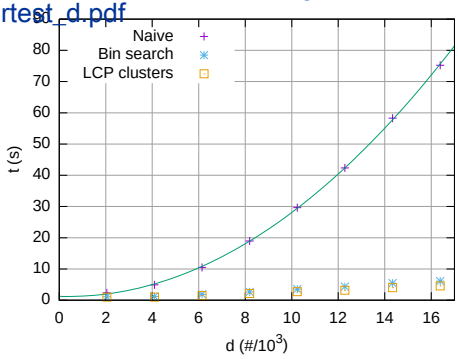
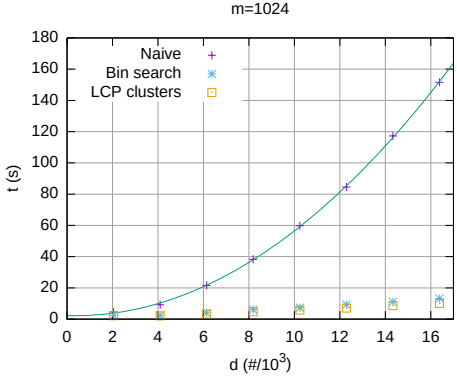


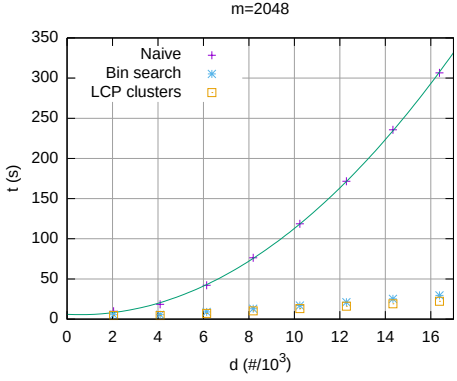
Figure3

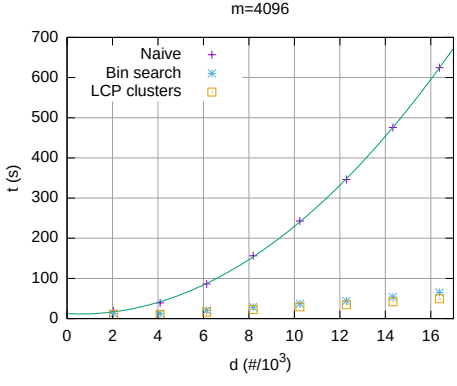
Click here to download Figure rtest_d.pdf











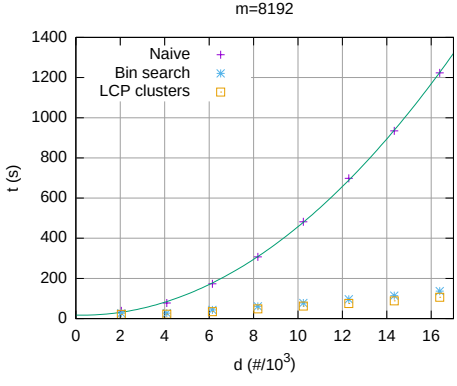


Figure4

Click here to download Figure tree.pdf



Personal Cover

Click here to download Personal Cover
amob_cover_letter_r1.pdf

Dear Editor,

Please find enclosed our research article entitled “Fast phylogenetic inference from typing data”.

The current ability to rapidly sequence whole microbial genomes, has the promise to revolutionize these fields by allowing the identification of thousands of potentially clinically relevant targets in the genome. NGS data can be used to detect outbreaks in hospital settings or in the food industry, e.g., by monitoring the spread of antimicrobial resistance, an ever-growing concern. It can help also in the development of vaccines by helping, for instance, to determine targets conserved in the entire bacterial population.

However, it is becoming clear that the bottleneck shifted from the production of sequence data to its analysis. One of the major challenges is on how phylogenetic inference methods can be scaled up to analyze thousands of genetic loci in thousands of isolates. Usually, computing genetic evolutionary distances among a set of typing profiles or taxa dominates the running time of many of these methods. It is important also to note that most of genetic evolution distance definitions rely, even if indirectly, on computing the pairwise Hamming distance among sequences or profiles.

In this work, we propose an average-case linear-time algorithm to compute pairwise Hamming distances among a set of taxa under a given Hamming distance threshold. This article includes both a theoretical analysis and extensive experimental results concerning the proposed algorithm. We further show how this algorithm can be successfully integrated into a well-known phylogenetic inference method, and how it can be used to speedup querying local phylogenetic patterns over large typing databases.

This work is an extension of the article “Towards distance-based phylogenetic inference in average-case linear-time”, presented at WABI 2017. It includes several revisions, including those raised by reviews of the workshop version of the paper, and it presents another real application of the developed methods, showing how it can be successfully used to speedup querying local phylogenetic patterns over large typing databases. The algorithm was integrated in INNUENDO platform, that is being developed under the INNUENDO project (http://www.innuendoweb.org/).

We wish to confirm that there are no known conflicts of interest associated with this publication.

The manuscript has been revised taking into account reviewers comments, and it has been read and approved by all named authors. Answers to reviewers’ comments are also provided with our resubmission.

With my best regards,

Alexandre Francisco