



Towards distance-based phylogenetic inference in average-case linear-time

Maxime Crochemore, Alexandre P Francisco, Solon P Pissis, Cátia Vaz

► To cite this version:

Maxime Crochemore, Alexandre P Francisco, Solon P Pissis, Cátia Vaz. Towards distance-based phylogenetic inference in average-case linear-time. International Workshop on Algorithms in Bioinformatics, Aug 2017, Boston, MA, United States. 10.4230/LIPIcs.WABI.2017.00 . hal-01616499

HAL Id: hal-01616499

<https://hal.science/hal-01616499>

Submitted on 13 Oct 2017

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.

Towards distance-based phylogenetic inference in average-case linear-time

Maxime Crochemore¹, Alexandre P. Francisco², Solon P. Pissis³,
and Cátia Vaz⁴

¹ Department of Informatics, King's College London, London, UK

² INESC-ID and Instituto Superior Técnico, Universidade de Lisboa, PT

³ Department of Informatics, King's College London, London, UK

⁴ INESC-ID and Instituto Superior de Engenharia de Lisboa, Instituto
Politécnico de Lisboa, PT

Abstract

Computing genetic evolution distances among a set of taxa dominates the running time of many phylogenetic inference methods. Most of genetic evolution distance definitions rely, even if indirectly, on computing the pairwise Hamming distance among sequences or profiles. We propose here an average-case linear-time algorithm to compute pairwise Hamming distances among a set of taxa under a given distance threshold. This paper 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 widely used phylogenetic inference method.

1998 ACM Subject Classification E.1 Data Structures, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases computational biology, phylogenetic inference, Hamming distance

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.00

1 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 molecular sequences or from profiles built by sequencing specific regions, *e.g.*, 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 foot-printing, to trace the origin and transmission of infectious diseases, or to study the co-evolution of hosts and parasites [10, 20].

In most cases, the process of phylogenetic inference starts with a multiple sequence alignment. Then tree-building methods are used. These methods rely on some distance-based analysis of sequences or profiles [21].

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 such distance between all pairs of sequences. The simplest approach is to use the Hamming distance, also called 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 [21, 10]. Although distance-based methods not always produce the best tree for the data, usually they also incorporate an optimality



© Maxime Crochemore, Alexandre P. Francisco, Solon P. Pissis and Cátia Vaz;
licensed under Creative Commons License CC-BY

17th Workshop on Algorithms in Bioinformatics (WABI 2017).

Editor: TBD; Article No. 00; pp. 00:1–00:12



Leibniz International Proceedings in Informatics

LIPIcs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

criterion into the distance model for getting more plausible phylogenetic reconstructions, such as the minimum evolution, least squares criterion [19] or the clonal complexes expansion and diversification [6].

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. A phylogenetic tree is obtained within this process. 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 then a common first step for distance-based methods, such as eBURST [7], goeBURST [8], Neighbor Joining [22] and UPGMA [23]. 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. However, depending on the underlying model of evolution and on the optimality criterion, it may not be necessary to be aware of the whole 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 [19]. The goeBURST, our use case, 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 $k > 0$.

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 $k > 0$, 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 [8], a widely used phylogenetic inference method.

Structure of the paper We describe and analyse the algorithm in Section 2. The goeBURST use case is presented in Section 3. The experimental evaluation on both synthetic and real data is presented in Section 4.

2 Closest pairs in linear time

Let P be the set of profiles (or sequences), defined over an integer alphabet Σ , (*i.e.*, $\Sigma = \{1, \dots, m^{O(1)}\}$), with $d = |P|$, $\sigma = |\Sigma|$ and all profiles of equal length m . Let also $H : P \times P \rightarrow \{0, \dots, m\}$ be the function such that $H(u, v)$ is the Hamming distance between profiles $u, v \in P$. Given a threshold $k > 0$, 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 the complete distance matrix for d profiles of length m .

We address this problem by indexing all profiles P in a suffix array (denoted by SA) and by computing the longest common prefix (denoted by LCP) array [15]. We rely also on a range minimum queries (RMQ) data structure [1, 2] over the LCP array (denoted by RMQ_{LCP}). The problem is then solved in three main steps:

1. Load and index all profiles in an SA.
2. Enumerate all candidate pairs given the maximum distance k .
3. Verify each candidate pair and check if the associated distance is no more than k .

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

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

Table 1 summarizes the data structures and strategies followed in each step. Profiles are concatenated and indexed in an 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 computing distances and enumerating candidate pairs faster. In what follows, we detail the above steps and show how data structures are used to improve the overall running time.

2.1 Step 1: Profile index

Profiles are concatenated and indexed in an SA S in $\mathcal{O}(md)$ time and space [13, 11].

Since we only need to compute the distances between profiles that have, at most, a dissimilarity of k , we can conceptually split each profile into k fragments of dimension $L = \lfloor \frac{m}{k+1} \rfloor$. It is then folklore knowledge that if two profiles are within distance k , they must share at least a fragment of length L . Our approach is based on the use of S to efficiently identify matching fragments among profiles. This permits to quickly identify candidate pairs.

2.2 Step 2: Candidate pairs enumeration

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

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 [12]. Since the SA is an ordered structure, for each contiguous suffixes $s_i s_{i+1} s_{i+2}$, with $0 \leq i < n - 2$, the common prefix between s_i and s_{i+1} is greater than the 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 or equal to L and they are aligned. This procedure can be done in $\mathcal{O}(kd^2)$ time in the *worst case*.

2.3 Step 3: Pairs verification

After getting the set of candidate pairs, a naïve solution is to verify the distance for each pair of profiles by comparing them in linear time, *i.e.*, $\mathcal{O}(m)$ time. However, if we compute the longest common prefix between each consecutive pair of elements within the SA, 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 . Since after a linear-time pre-processing over the LCP array, RMQ can be answered in constant time per query [1], we obtain a faster approach. This alternative approach takes $\mathcal{O}(k)$ time to verify each candidate pair instead of $\mathcal{O}(m)$ time.

2.4 Analysis

Algorithm 1 below details the solution based on LCP clusters and Theorem 1 shows that this algorithm runs in linear time and linear space on average. We rely here on well known results concerning the linear time construction of the SA [13, 11] and the LCP array [12], as well as the RMQ pre-processing in linear time and space [2].

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

Aligned(i) Let $\ell = i \bmod m$, *i.e.*, the starting position of the suffix s_i within a profile.

Then this subroutine returns ℓ/L if ℓ is multiple of L , and -1 otherwise.

HD(p_i, p_j, ℓ) Given two profiles which share a substring of length L , starting at index ℓL , 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 stop 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 distance $k < m$, Algorithm 1 runs in average-case time $\mathcal{O}(md)$ and space $\mathcal{O}(md)$ if*

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

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

Let B denote the number of blocks over x , and let L denote the block length. We set

$$L = \lfloor \frac{m}{k+1} \rfloor, \quad B = \lfloor \frac{md}{L} \rfloor.$$

By P we denote a maximal set of indices of the LCP table for x satisfying: (i) the length of the longest common prefix between any two suffixes starting at these indices is at least L ; (ii) both of these suffixes start at the starting position of a block; (iii) and both indices correspond to the starting position of the i th block in their profiles. Processing all such sets P requires time

$$\text{VER}_{i,j} \cdot \text{Occ}$$

where $\text{VER}_{i,j}$ is the time required to process a pair i, j of elements of a set P , and Occ is the sum of $|P|^2$ over all such sets P . We have that $\text{VER}_{i,j} = \mathcal{O}(k)$. Additionally, by the stated assumption on the d profiles, the expected value for Occ is no more than $\frac{Bd}{\sigma^L}$: we have B blocks and each block can only match at most d other blocks. Hence, the algorithm requires on average the running time

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

Since $m \geq k + 1$ by hypothesis, we have the following:

$$\begin{aligned} k \cdot \frac{Bd}{\sigma^L} &= \frac{k \cdot \lfloor \frac{md}{\lfloor \frac{m}{k+1} \rfloor} \rfloor \cdot d}{\sigma^{\lfloor \frac{m}{k+1} \rfloor}} \leq \frac{k \cdot (\frac{md}{m/(k+1)-1}) \cdot d}{\sigma^{\frac{m}{k+1}-1}} \leq \frac{(md)^2}{(md)^{\frac{\log \sigma}{\log md} (\frac{m}{k+1}-1)}} \\ &= (md)^{2 - \frac{(m-k-1) \log \sigma}{(k+1) \log md}}. \end{aligned}$$

Algorithm 1: Algorithm using LCP clusters.

```

1 Input: A set  $P$  of  $d$  profiles of length  $m$ ; a threshold  $k > 0$ 
2 Output: The set  $X$  of distinct pairs of profiles that are at most at distance  $k$ , i.e.,
    $X = \{(u, v) \in P \times P \mid u < v \text{ and } H(u, v) \leq k\}$ .
3 Initialization: Let  $x = x[0 \dots n - 1]$  be the string of length  $n = md$  obtained after
   concatenating together the  $d$  profiles. Construct an SA  $\mathcal{S}$  for  $x$ , the LCP array for  $\mathcal{S}$ 
   and pre-process  $\text{RMQ}_{\text{LCP}}$ . Initialize a hash table  $H$  to track verified pairs.
4 Candidate pairs enumeration:
5  $X := \emptyset$ ;  $\ell_p = -1$ ;  $C_t := \emptyset$ , for  $0 \leq t < k$ 
6 foreach  $1 \leq i < n$  do
7    $\ell := \text{LCP}[i]$ 
8   if  $\ell \geq L$  then
9      $p_i := \lfloor \mathcal{S}[i]/m \rfloor$ 
10     $x := \text{Aligned}(i)$ 
11    if  $x \neq -1$  then
12       $C_x := C_x \cup \{p_i\}$ 
13    if  $\ell_p = -1$  then
14       $p_{i-1} := \lfloor \mathcal{S}[i-1]/m \rfloor$ 
15       $x := \text{Aligned}(i)$ 
16      if  $x \neq -1$  then
17         $C_x := C_x \cup \{p_{i-1}\}$ 
18     $\ell_p := \ell$ 
19  else if  $\ell_p \neq -1$  then
20    Pairs enumeration:
21    foreach  $C_t$ , with  $0 \leq t < k$  do
22      foreach  $(p, q) \in C_t \times C_t$  such that  $p < q$  do
23        if  $(p, q) \notin H$  then
24           $H := H \cup \{(p, q)\}$ 
25           $\delta := \text{HD}(p, q, t)$ 
26          if  $\delta \leq k$  then
27             $X := X \cup \{(p, q)\}$ 
28     $\ell_p = -1$ ;  $C_t := \emptyset$ , for  $0 \leq t < k$ 
29 Finalize: Return the set  $X$ .

```

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 $\mathcal{O}(md)$. ◀

3 Use case

The computation of a distance matrix is a main step in distance-based methods for phylogenetic inference. This step dominates however 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 pairs. But for some methods, or when we are only interested on local phylogenies, one does not need to know all pairwise distances for reconstructing a phylogenetic tree. The problem addressed in this paper was motivated by the goeBURST algorithm, our use case. goeBURST is one of such methods for which one must know only the pairs of sequences that are at distance at most $k > 0$. The solution proposed here can however be extended to other distance-based phylogenetic inference algorithms, namely those that rely directly or indirectly on the computation of Hamming distance.

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 the founder — 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 [8]. Given $k > 0$, we can define a graph $G = (V, E)$, where $V = P$ 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 iteratively selecting links connecting profiles in different trees and with the higher number of SLVs. In the current implementation it is implicitly defined a total order for links 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 [14], where each sequence is a node and the link weights are defined by the total order. By construction, the pairs at distance k will be joined before the pairs at distance $k + 1$, with $k \geq 0$.

4 Experimental evaluation

We evaluated the proposed approach on both real and synthetic data. We used real datasets obtained through different typing schemas, namely wide-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. Simulated data comprises 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 <https://github.com/y-256/libdivsufsort> and <http://www.larsson.dogma.net/qsufsort.c> libraries, respectively. For RMQ over the LCP array, we adopt a fast well-known solution that uses constant time per query and linearithmic space for pre-processing [1].

All tests were conducted on a machine running Linux, with an Intel(R) Xeon(R) CPU

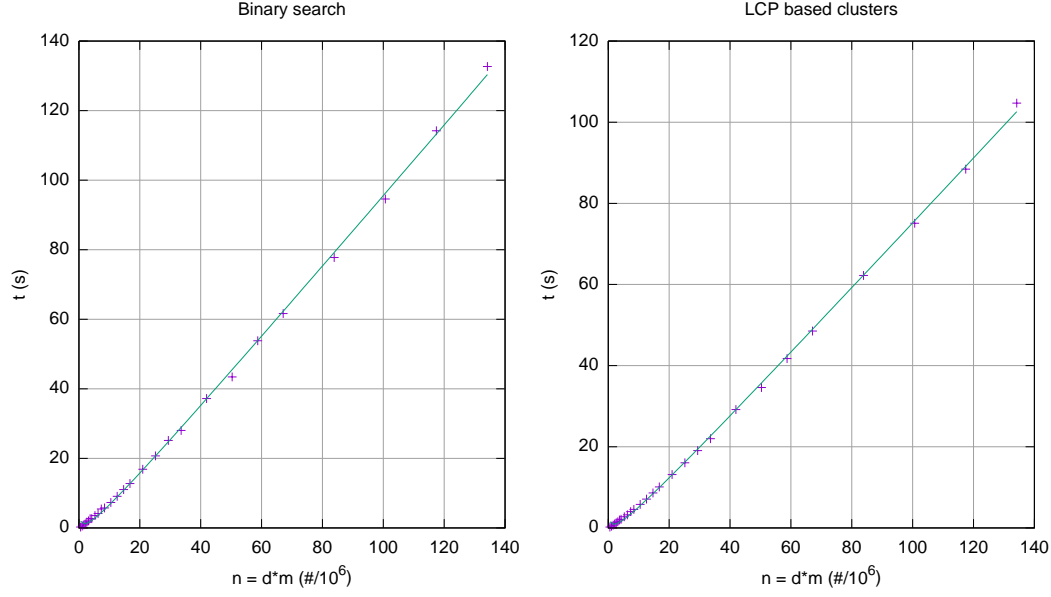


Figure 1 Synthetic data, with $\sigma = 2$ and $k = \lfloor m/(2 \log m) \rfloor$ according to Theorem 1. Running time for computing pairwise distances by finding lower and upper bounds in the SA, and by processing LCP based clusters, as function of the input size n .

E5-2630 v3 @ 2.40GHz (8 cores, cache 32KB/4096KB) and with 32GB of RAM. All binaries were produced using GCC 5.3 with full optimization enabled.

4.1 Synthetic data

We 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 slight above linear. Since we included the time for constructing the SA, the LCP array and the RMQ index, with the last one in linearithmic time, that was expected.

We tested also for k above the bound in Theorem 1. For $d = m = 4096$ and a binary alphabet, the bound for k given in Theorem 1 is at most $170 = \lfloor m/(2 \log m) \rfloor$. 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 upper bounds in the SA, and by processing LCP based clusters becomes worse 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 upper bounds in the SA, and by processing LCP based clusters. Hence, for synthetic data, as described by Theorem 1, the result holds.

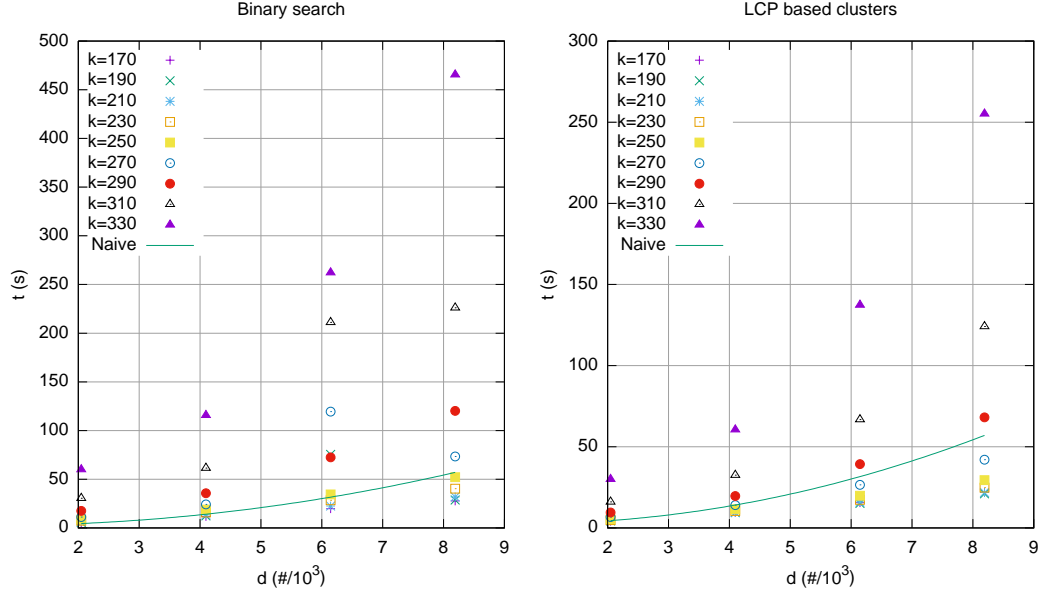


Figure 2 Synthetic data, with $\sigma = 2$ and $m = 4096$. Running time for computing pairwise distances by finding lower and upper bounds in the SA, and by processing LCP based clusters, as function of the number d of profiles and for different threshold k .

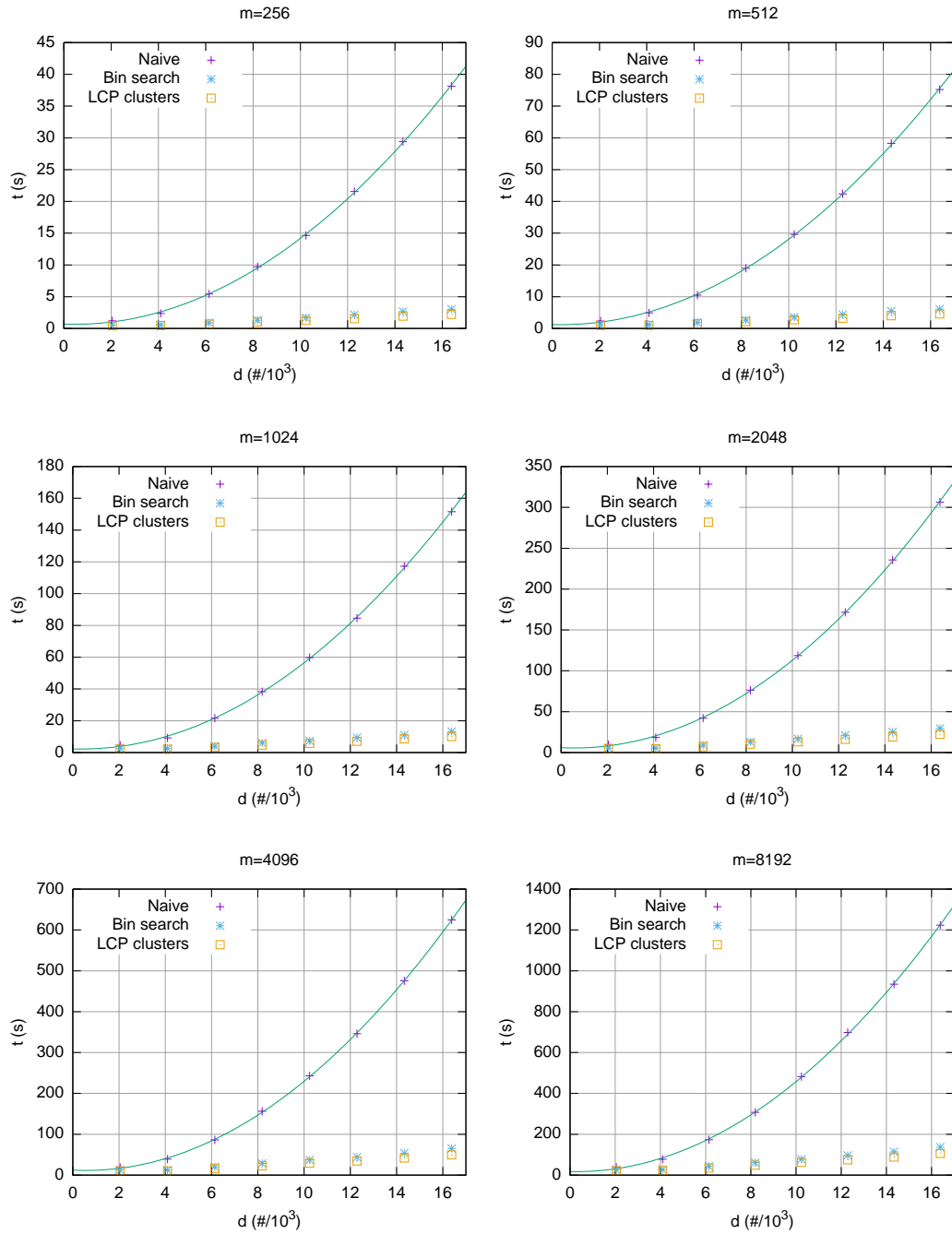
Table 2 Real datasets used in the experimental evaluation. (*)Dataset provided by the Molecular Microbiology and Infection Unit, IMM.

Dataset	Typing method	Profile length	Number of distinct elements	Reference
<i>Campylobacter jejuni</i>	wgMLST	5446	5669	(*)
<i>Salmonella enterica</i>	wgMLST	3002	6861	[5]
<i>Salmonella typhi</i>	SNP	22143	1534	[18]
<i>Streptococcus pneumoniae</i>	cgMLST	235	1968	[4, 3, 17]

4.2 Real datasets

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

In most cases, the approach based on the LCP clusters is the fastest up to three orders of magnitude compared to the naïve approach. As expected by Theorem 1, in the case when real data are not uniformly random, our method works reasonably well for smaller values of k than the ones implied by the condition. As an example, the upper bound on k for *C. jejuni* would be larger than 219, but the running time for the naïve approach is already better for $k = 64$. We should note however that the number of candidate pairs at distance at most k is much higher than the expected number if data is uniformly random. This tells us that we can design a simple hybrid algorithm 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 paper.



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

■ **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 (%)
<i>C. jejuni</i>	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
<i>S. enterica</i>	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
<i>S. typhi</i>	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
<i>S. pneumoniae</i>	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

We incorporated the approach based on finding lower and upper bounds in the SA in the implementation of goeBURST algorithm, discussed in Section 3. 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 the *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 Section 3, 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 [8]. 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. Note that this tree is optimal with respect to the criterion used by the goeBURST algorithm. Comparing this tree with other inference methods is beyond the scope of this article; the focus here is on the faster computation of an optimal tree under this model. The nodes are colored according to one of the loci for which profiles in this cluster differ. 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 [9].

References

- 1 Michael A. Bender and Martín Farach-Colton. The LCA problem revisited. In *LATIN 2000: Theoretical Informatics: 4th Latin American Symposium*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000.

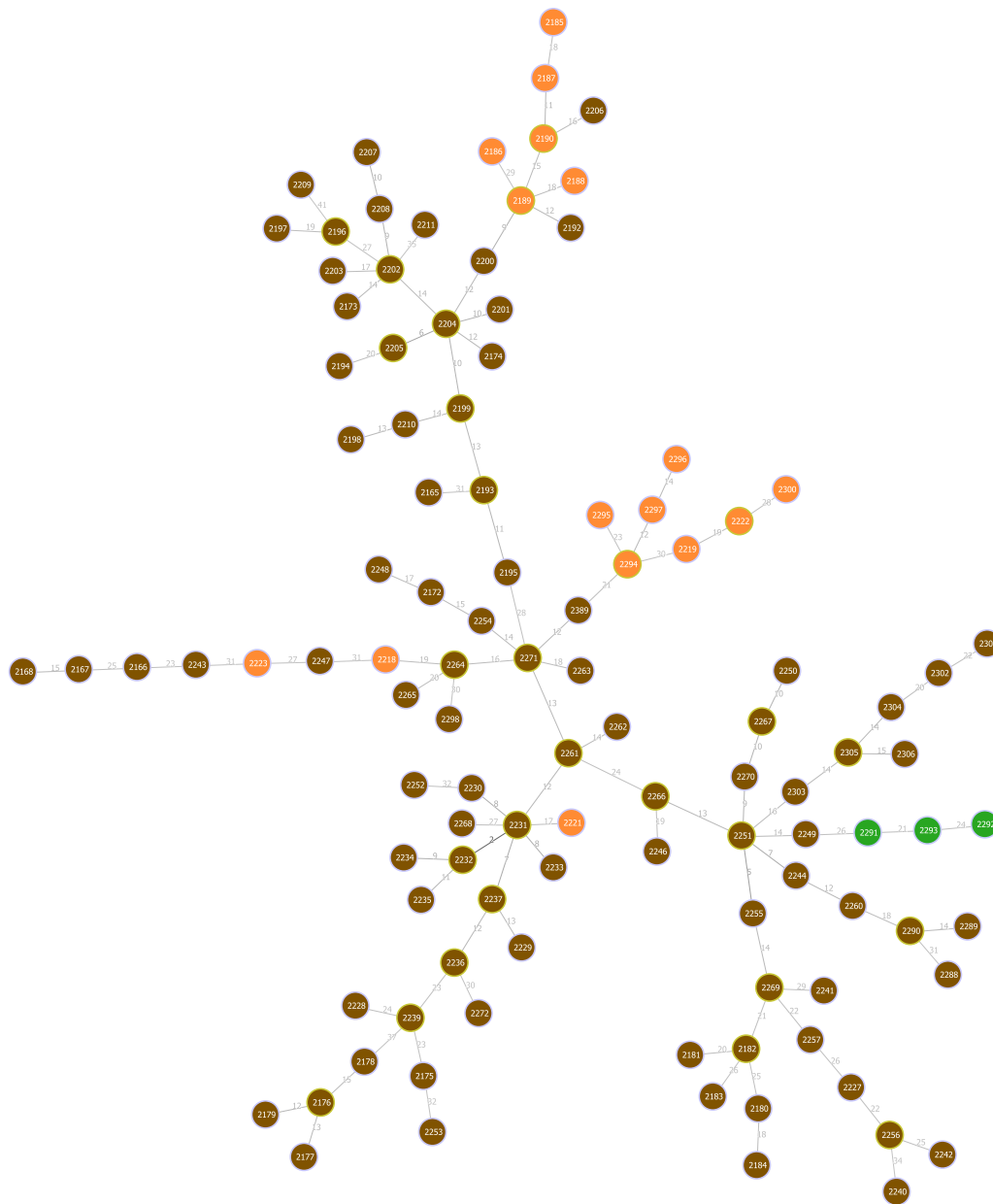


Figure 4 The tree inferred for the largest connected component found with $k = 52$ for the *C. jejuni* dataset. Image produced with PHYLOViZ [16].

- 2 Michael A Bender, Martín Farach-Colton, Giridhar Pemmasani, Steven Skiena, and Pavel Sumazin. Lowest common ancestors in trees and directed acyclic graphs. *Journal of Algorithms*, 57(2):75–94, 2005.
- 3 Claire Chewapreecha, Simon R Harris, Nicholas J Croucher, Claudia Turner, Pekka Marttinen, Lu Cheng, Alberto Pessia, David M Aanensen, Alison E Mather, Andrew J Page, Susannah J Salter, David Harris, Francois Nosten, David Goldblatt, Jukka Corander, Julian Parkhill, Paul Turner, and Stephen D Bentley. Dense genomic sampling identifies highways of pneumococcal recombination. *Nature Genetics*, 46(3):305–309, 2014.
- 4 Nicholas J Croucher, Jonathan A Finkelstein, Stephen I Pelton, Patrick K Mitchell,

- Grace M Lee, Julian Parkhill, Stephen D Bentley, William P Hanage, and Marc Lipsitch. Population genomics of post-vaccine changes in pneumococcal epidemiology. *Nature Genetics*, 45(6):656–663, 2013.
- 5 EnteroBase. Enterobase.warwick.ac.uk. URL: <http://enterobase.warwick.ac.uk>.
 - 6 Edward J Feil, Edward C Holmes, Debra E Bessen, Man-Suen Chan, Nicholas PJ Day, Mark C Enright, Richard Goldstein, Derek W Hood, Awdhesh Kalia, Catrin E Moore, 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.
 - 7 Edward J Feil, Bao C Li, David M Aanensen, William P Hanage, and Brian G Spratt. 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.
 - 8 Alexandre P Francisco, Miguel Bugalho, Mário Ramirez, and João Carriço. Global optimal ebust analysis of multilocus typing data using a graphic matroid approach. *BMC bioinformatics*, 10(1), 2009.
 - 9 Alexandre P Francisco, Cátia Vaz, Pedro T Monteiro, José Melo-Cristino, Mário Ramirez, and Joao A Carriço. PHYLOViZ: phylogenetic inference and data visualization for sequence based typing methods. *BMC bioinformatics*, 13(1):87, 2012.
 - 10 Daniel H Huson, Regula Rupp, and Celine Scornavacca. *Phylogenetic networks: concepts, algorithms and applications*. Cambridge University Press, 2010.
 - 11 Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. Linear work suffix array construction. *Journal of ACM*, 53(6):918–936, 2006.
 - 12 Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Annual Symposium on Combinatorial Pattern Matching*, pages 181–192. Springer, 2001.
 - 13 Pang Ko and Srinivas Aluru. Space efficient linear time construction of suffix arrays. In *Annual Symposium on Combinatorial Pattern Matching*, volume 2676 of *Lecture Notes in Computer Science*, pages 200–210. Springer, 2003.
 - 14 Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956.
 - 15 Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.
 - 16 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*, 33(1):128–129, 2017.
 - 17 National Center for Biotechnology Information. GeneBank. URL: ftp://ftp.ncbi.nih.gov/genomes/archive/old_genbank/Bacteria/.
 - 18 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.
 - 19 Fabio Pardi and Olivier Gascuel. Distance-based methods in phylogenetics. In *Encyclopedia of Evolutionary Biology*, pages 458–465. Elsevier, 2016.
 - 20 D Ashley Robinson, Edward J Feil, and Daniel Falush. *Bacterial population genetics in infectious disease*. John Wiley & Sons, 2010.
 - 21 Naruya Saitou. *Introduction to evolutionary genomics*. Springer, 2013.
 - 22 Naruya Saitou and Masatoshi Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425, 1987.
 - 23 Robert R Sokal. A statistical method for evaluating systematic relationships. *Univ Kans Sci Bull*, 38:1409–1438, 1958.