# Towards Distance-Based Phylogenetic Inference in Average-Case Linear-Time*

**Maxime Crochemore[1], Alexandre P. Francisco[2], Solon P. Pissis[3], and Cátia Vaz[4]**

1 **Department of Informatics, King's College London, London, UK**
2 **INESC-ID and Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal**
3 **Department of Informatics, King's College London, London, UK**
4 **INESC-ID and Instituto Superior de Engenharia de Lisboa, Instituto Politécnico de Lisboa, Lisbon, Portugal**

## 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 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.

**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.2017.9

## 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 [11, 23].

In most cases, the process of phylogenetic inference starts with a multiple alignment of the sequences under study; and then tree-building methods are used. These methods rely on some distance-based analysis of sequences or profiles [24].

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

* This work was partly supported by the Royal Society International Exchanges Scheme, and by national funds through FCT – Fundação para a Ciência e Tecnologia, under projects BacGenTrack (TUBITACK/0004/2014), PRECISE (SAICTPAC/0021/2015) and UID/CEC/500021/2013.


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 [11, 24]. 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 [22] or the clonal complexes expansion and diversification [7].

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 a common first step for distance-based methods, such as eBURST [8], goeBURST [9], Neighbor Joining [25] and UPGMA [26]. 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 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 [22]. The goeBURST, our use case 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 $\sigma$ 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.

**Structure of the article.** We describe and analyze the proposed algorithm in Section 2. The goeBURST use case is presented in Section 3. The experimental evaluation using both synthetic and real datasets is presented in Section 4.

## 2 Closest pairs in linear time

Let $P$ be the set of profiles (or sequences) each of length $m$, defined over an integer alphabet $\Sigma$, (*i.e.*, $\Sigma = \{1, \ldots, m^{O(1)}\}$), with $d = |P|$ and $\sigma = |\Sigma|$. Let also $H : P \times P \to \{0, \ldots, m\}$ 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 $m$.

**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 | $\text{RMQ}_{\text{LCP}}$ |

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 [1, 2] over the LCP array (denoted by $\text{RMQ}_{\text{LCP}}$). 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.

## 2.1 Step 1: Profile indexing

Profiles are concatenated and indexed in an SA in $\mathcal{O}(md)$ time and space [12, 14]. 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.

## 2.2 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 [13]. 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.

## 2.3 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 [1], 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.

## 2.4 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 [12, 14] and the LCP array [13], as well as the linear-time pre-processing for the RMQ data structure [2].

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 $\mathrm{RMQ}_{\mathrm{LCP}}(i, j)$ returns the index of the smallest element in the subarray LCP[$i \dots j$] in constant time [2]. 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 $\ell$ is multiple of $\mathcal{L}$, and $-1$ otherwise.

**HD($p_i, p_j, \ell$).** 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 $\mathrm{RMQ}_{\mathrm{LCP}}$ 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 $\Sigma$ of size $\sigma > 1$ with the letters of the profiles being independent and identically distributed random variables uniformly distributed over $\Sigma$, 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 $\mathcal{L}$;
2. both of these suffixes start at the starting position of a block;
3. and both indices correspond to the starting position of the $i$th 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

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

**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 $s = s[0 \dots n-1]$ 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 $\text{RMQ}_{\text{LCP}}$. Initialize a hash table $H$ 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**
  $\ell := \text{LCP}[i]$
  **if** $\ell \geq \mathcal{L}$ **then**
    $p_i := \lfloor \mathcal{S}[i]/m \rfloor$
    $x := \text{Aligned}(i)$
    **if** $x \neq -1$ **then**
      $C_x := C_x \cup \{p_i\}$
    **if** $\ell_p = -1$ **then**
      $p_{i-1} := \lfloor \mathcal{S}[i-1]/m \rfloor$
      $x := \text{Aligned}(i-1)$
      **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** $(p, q) \notin H$ **then**
          $H := H \cup \{(p, q)\}$
          $\delta := \text{HD}(p, q, t)$
          **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 $X$.

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 $\mathcal{O}(md)$. ◀

## 3 Use case: 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 [9], our use case. 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 [11, 24]. In both cases we must start by computing the Hamming distance among sequences, but not necessarily all of them [22].

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 [21]. 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 [15]. 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 [15], where each sequence is a node and the link weights are defined by the total order. By construction, the pairs at distance $\delta$ will be joined before the pairs at distance $\delta + 1$.

## 4 Experimental evaluation

We evaluated the proposed approach using both real and synthetic datasets. 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. 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 `https://github.com/y-256/libdivsufsort` and `http://www.larsson.dogma.net/qsufsort.c` libraries, 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 [1].

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.

### 4.1 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 upper 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

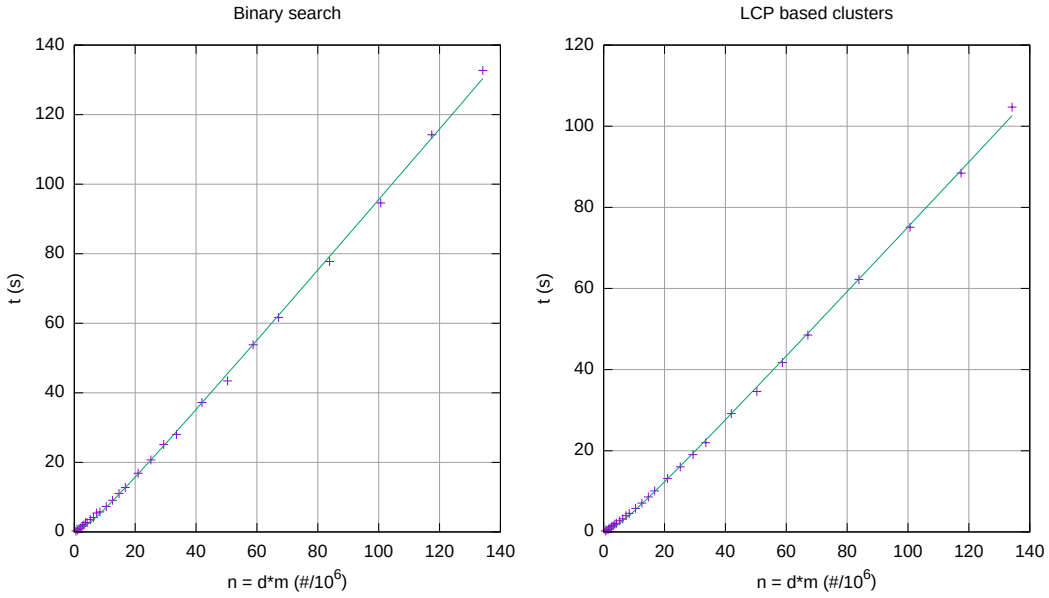


**Figure 1** Synthetic datasets, with $\sigma = 2$ and $k = \lfloor m/(2 \log m) \rfloor$ according to Theorem 1. Running time for computing pairwise distances by finding lower and upper bounds in the SA, and by processing LCP based clusters, as function of the input size $n = dm$.

**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 | [6] |
| *Salmonella typhi* | SNP | 22143 | 1534 | [20] |
| *Streptococcus pneumoniae* | cgMLST | 235 | 1968 | [4, 3, 19] |

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.

## 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 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

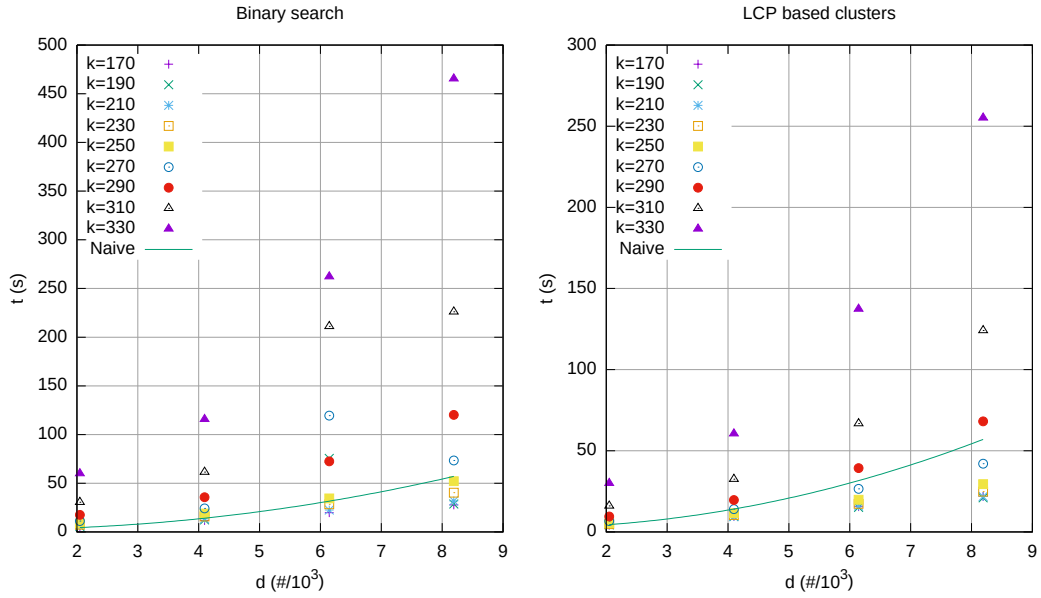


**Figure 2** Synthetic datasets, with $\sigma = 2$ and $m = 4096$. Running time for computing pairwise distances by finding lower and upper bounds in the SA, and by processing LCP based clusters, as function of the number $d$ of profiles and for different values of $k$.

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 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 *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 [9]. 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 [21]. 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.

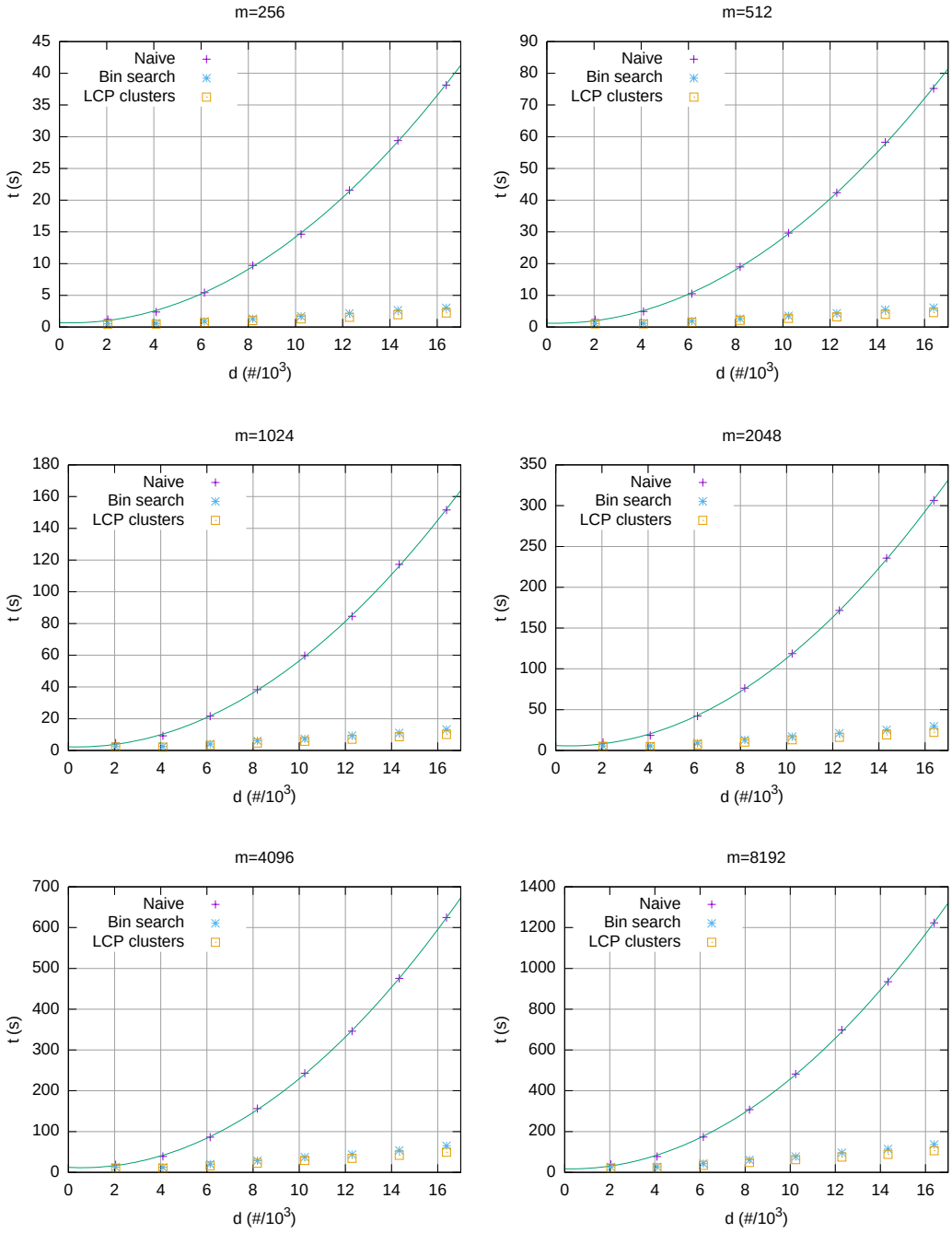


**Figure 3** Synthetic datasets, with $\sigma = 2$ and $k = \lfloor m/(2 \log m) \rfloor$ according to Theorem 1. Running time for computing pairwise distances naïvely, by finding lower and upper bounds in the SA, and by processing LCP based clusters, as a function of the number $d$ of profiles.

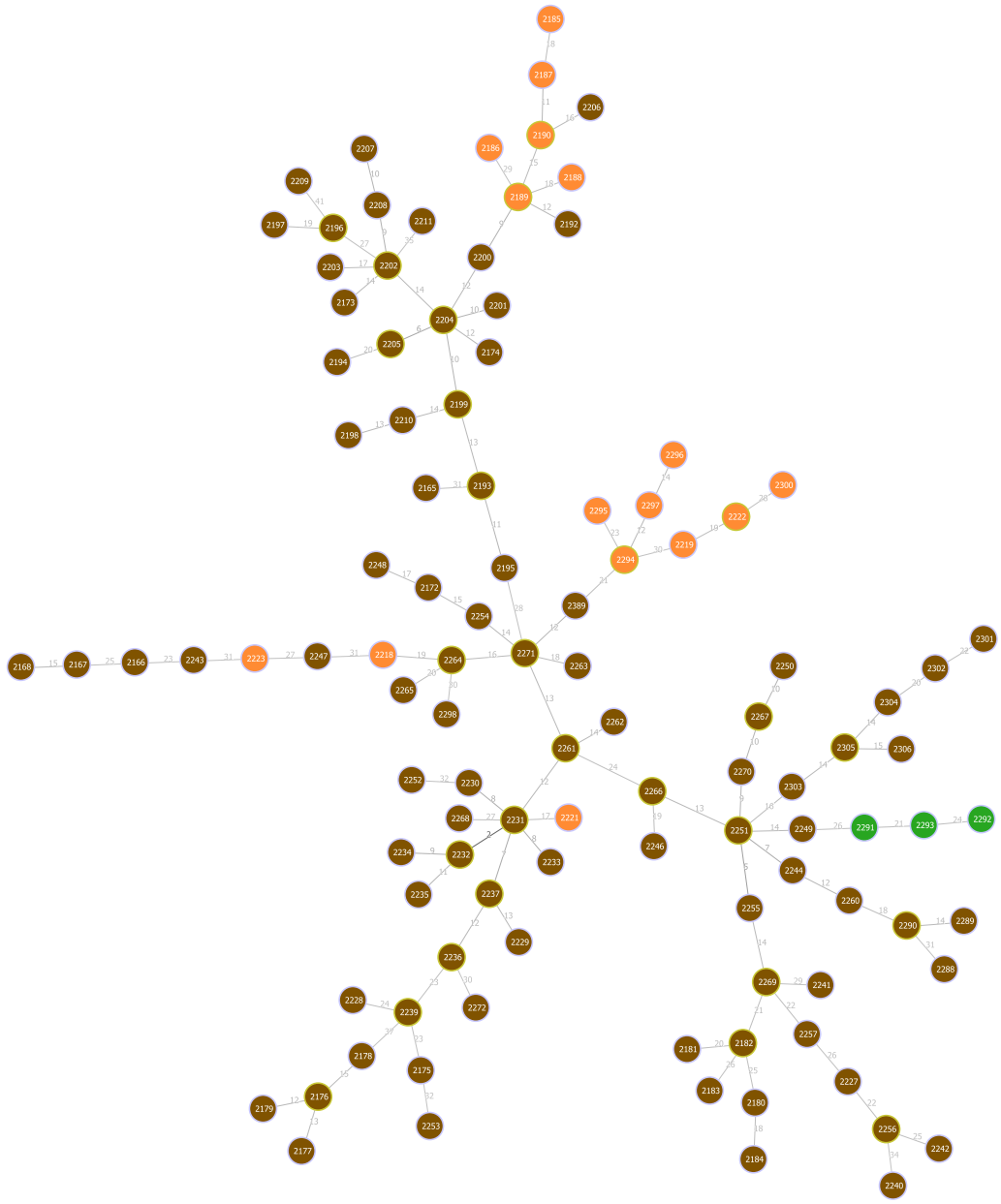


**Figure 4** The tree inferred for the largest connected component found with $k = 52$ for the *C. jejuni* dataset. Image produced by PHYLOViZ [18].

**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 |

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 [10].

## 5 Concluding remarks

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 [22].

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 [9], which relies on a truncated distance matrix by construction. 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 $\sigma$ 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 [24].

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. 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 more 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 [17]. This and other challenges raised above are left as future work.

## 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. doi:10.1007/10719839_9.

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. doi:10.1016/j.jalgor.2005.08.001.

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. doi:10.1038/ng.2895.

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. doi:10.1038/ng.2625.

5 Richard Desper and Olivier Gascuel. 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 EnteroBase. Enterobase.warwick.ac.uk. URL: http://enterobase.warwick.ac.uk.

7 Edward J. Feil, Edward C. Holmes, Debra E. Bessen, Man-Suen Chan, Nicholas P. J. 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. doi:10.1073/pnas.98.1.182.

8 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. doi:10.1128/JB.186.5.1518-1530.2004.

9 Alexandre P Francisco, Miguel Bugalho, Mário Ramirez, and João Carriço. 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.

10 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. doi:10.1186/1471-2105-13-87.

11 Daniel H. Huson, Regula Rupp, and Celine Scornavacca. *Phylogenetic networks: concepts, algorithms and applications*. Cambridge University Press, 2010. doi:10.1017/CBO9780511974076.

12 Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. Linear work suffix array construction. *Journal of ACM*, 53(6):918–936, 2006. doi:10.1145/1217856.1217858.

13 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. doi:10.1007/3-540-48194-X.

14 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. doi:10.1016/j.jda.2004.08.002.

15 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. doi:10.2307/2033241.

16 Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993. doi:10.1137/0222058.

17 J. Ian Munro, Yakov Nekrich, and Jeffrey Scott Vitter. Dynamic data structures for document collections and graphs. In *Proceedings of the 34th ACM Symposium on Principles of Database Systems*, pages 277–289. ACM, 2015. doi:10.1145/2745754.2745778.

18 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. doi:10.1093/bioinformatics/btw582.

19 National Center for Biotechnology Information. GeneBank. URL: ftp://ftp.ncbi.nih.gov/genomes/archive/old_genbank/Bacteria/.

20 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. doi:10.1099/mgen.0.000056.

21 Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., 1982.

22 Fabio Pardi and Olivier Gascuel. Distance-based methods in phylogenetics. In *Encyclopedia of Evolutionary Biology*, pages 458–465. Elsevier, 2016. doi:10.1016/B978-0-12-800049-6.00206-7.

23 D. Ashley Robinson, Edward J. Feil, and Daniel Falush. *Bacterial population genetics in infectious disease*. John Wiley & Sons, 2010. doi:10.1002/9780470600122.

24 Naruya Saitou. *Introduction to evolutionary genomics*. Springer, 2013. doi:10.1007/978-1-4471-5304-7.

25 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. doi:10.1093/oxfordjournals.molbev.a040454.

26 Robert R. Sokal. A statistical method for evaluating systematic relationships. *Univ Kans Sci Bull*, 38:1409–1438, 1958.