Grossi *et al. Algorithms Mol Biol (2016) 11:12*
DOI 10.1186/s13015-016-0076-6

Algorithms for Molecular Biology

RESEARCH Open Access

Circular sequence comparison: algorithms and applications

Roberto Grossi[1,2], Costas S. Iliopoulos[3], Robert Mercas[3,4], Nadia Pisanti[1,2], Solon P. Pissis[3*], Ahmad Retha[3] and Fatima Vayani[3]

Abstract

Background: Sequence comparison is a fundamental step in many important tasks in bioinformatics; from phylogenetic reconstruction to the reconstruction of genomes. Traditional algorithms for measuring approximation in sequence comparison are based on the notions of distance or similarity, and are generally computed through sequence alignment techniques. As *circular* molecular structure is a common phenomenon in nature, a caveat of the adaptation of alignment techniques for circular sequence comparison is that they are computationally expensive, requiring from super-quadratic to cubic time in the length of the sequences.

Results: In this paper, we introduce a new distance measure based on q-grams, and show how it can be applied effectively and computed efficiently for circular sequence comparison. Experimental results, using real DNA, RNA, and protein sequences as well as synthetic data, demonstrate orders-of-magnitude superiority of our approach in terms of efficiency, while maintaining an accuracy very competitive to the state of the art.

Background

Biological motivation

Circular molecular structures are present, in abundance, in all domains of life: bacteria, archaea, and eukaryotes; and in viruses. They can be composed of either amino or nucleic acids. The following is an overview of such occurrences, and exhaustive reviews can be found in [1] (proteins) and [2] (DNA).

Double-stranded, circular chromosomes and plasmids are found in most bacteria and archaea. Whole-genome comparison is a very useful tool in classifying bacterial strains, as well as inferring phylogenetic associations between them. This is due to the dense structure of bacterial chromosomes, caused by the absence of introns, and the organisation of genes into operons. The extended benefit of aligning plasmids is the ability to identify important genes, such as antibiotic resistance genes, thereby enabling their study and exploitation by genetic engineering techniques [3].

The most familiar examples of such structures in eukaryotes are mitochondrial (MtDNA) and plastid DNA. MtDNA is, in most cases, inherited solely from the mother, and so is generally conserved. Human MtDNA is double-stranded, with a length of 16,569 base pairs (bp), consisting of just 37 genes encoding 13 proteins and 24 RNA molecules [4]. The absence of recombination in these sequences allows them to be used as simple indicators of phylogenetic evolution, and their high mutation rate is a powerful discriminative feature [5, 6]. There also exist smaller structures, called extrachromosomal circular DNA, which are similar to plasmids in bacterial cells. They are described as one of the characteristics of genomic plasticity in eukaryotes [7] and may derive from MtDNA [8].

It is common knowledge that many viral genomes are circular. Viral genomes vary greatly in size and structure. They can be made up of either RNA or DNA, and can be single- or double-stranded. Multiple sequence alignment of viral genomes can be useful in the elucidation of novel sites of interest [9], as well as the inference of evolutionary relationships [10]. This is particularly important in studying their pathogenicity, due to the rapid rate of mutation of viruses. Viroids are plant pathogens

*Correspondence: solon.pissis@kcl.ac.uk
[3] Department of Informatics, King's College London, London, UK
Full list of author information is available at the end of the article

that comprise very small, single-stranded, circular RNA. Their multiple sequence alignment could prove useful in the analysis of their secondary structures and, therefore, the mechanisms by which they infect host plant cells [11].

Naturally-occuring circular proteins are found in both prokaryotes and eukaryotes [1]. Bacteriocins are very small toxins produced by bacteria in order to compete with closely-related bacterial strains. Many of these are circular, including gassericin A, found in *Lactobacillus gasseri* LA39 [12], and circularin A, found in *Clostridium beijerinckii* [13]. An interesting phenomenon known to occur naturally in linear protein structures is circular permutation [14]. This can be exemplified by swaposins: proteins highly-similar to saposins, resulting from circularly permuted linear peptide sequences [15]. The ability to align linear sequences from circular proteins can significantly speed up and enhance their analyses, and could also lead to the discovery of novel pairs of circularly permuted proteins.

Our problem

Conventional tools, designed for linear sequences, could yield an incorrectly high genetic distance between closely related circular sequences. Indeed, when sequencing molecules, the position where a circular sequence starts can be totally arbitrary. Due to this arbitrariness, a suitable rotation of one sequence would give much better results for a pairwise alignment, and hence highlight a similarity that any linear alignment would miss. A practical example of the benefit this can bring to sequence analysis is the following. Linearized human (NC_001807) and chimpanzee (NC_001643) MtDNA sequences, obtained from GenBank [16], do not start in the same region. Their pairwise sequence alignment using EMBOSS Needle [17] (default parameters) gives a similarity of 85.1 % and consists of 1195 gaps. However, taking different rotations of these sequences into account yields a much more significant alignment with a similarity of 91 % and only 77 gaps. This example motivates the design of efficient algorithms that are specifically devoted to the comparison of circular sequences [18–20].

In this paper, we consider the pairwise circular sequence comparison problem. Under the edit distance model, it consists in finding an optimal linear alignment of two circular strings. This problem, for two strings x and y of length m and $n \geq m$, respectively, can be solved under the edit distance model in time $\mathcal{O}(nm \log m)$ [21]. Several other super-quadratic [22] and *approximate* quadratic-time [23] algorithms exist. Trivially, for molecular biology applications, the same problem can be solved in time $\mathcal{O}(nm^2)$, if extending the problem with scoring matrices and affine gap penalty scores. A *direct* application of pairwise circular sequence comparison is progressive multiple circular sequence alignment [11, 24, 25]. Multiple circular sequence alignment has also been considered in [26] under the Hamming distance model.

To the best of our knowledge, there is no fast (that is, with sub-quadratic time complexity) and exact (or at least very accurate) algorithm for circular sequence comparison under some realistic model (that is, allowing *indels*). Taking into account edit distance rather than Hamming distance is computationally challenging as the search space for seeking similarity is wider. Algorithms that speed up the process of string matching, by filtering out candidate positions in which a particular string can never occur, are known as filters. Filters that work for Hamming distance do not work in general for edit distance [27] as well. An exception to this are the q-gram filtering techniques [28] that have successfully been used for string matching under the edit distance model (e.g. [29–31]), as well as for multiple local alignments, both under the Hamming [32] and edit [31] distance models.

Our contribution

We present new efficient q-gram-based methods for pairwise circular sequence comparison. Specifically, our contribution is threefold.

1. We introduce the *β-blockwise q-gram distance* between two strings x and y, that is, a more powerful generalization of the q-gram distance introduced as a string distance measure in [28]. Intuitively, and similarly to [29–31], this generalization comprises partitioning x and y in β *blocks* each, as evenly as possible, computing the q-gram distance between the corresponding block pairs, and then summing up the distances computed blockwise.
2. We present an algorithm based on the suffix array [33] that finds the rotation of x such that the β-blockwise q-gram distance between the rotated x and y is minimal, in time and space $\mathcal{O}(\beta m + n)$, where $m = |x|$ and $n = |y|$, thereby solving *exactly* the circular sequence comparison problem under the β-blockwise q-gram distance measure. We also present a simple heuristic algorithm to solve an *approximate* version of the problem.
3. We present an experimental study, using real and synthetic data, which demonstrates orders-of-magnitude superiority of our approach, in terms of efficiency, while maintaining an accuracy very competitive to the *optimal* obtained after considering all rotations of x against y using EMBOSS Needle.

The paper is organized as follows. "Definitions and properties" section gives some preliminary definitions, notation, and properties. "Algorithms" section describes two algorithms, one is a heuristic approach and the other

is an exact algorithm for circular sequence comparison under the β-blockwise q-gram distance measure. "Implementation" section provides details of the implementation of the algorithms. "Experimental results" section presents the experimental results of the performance and accuracy of the algorithms. Finally, "Conclusions" section gives some concluding remarks and future proposals. A preliminary version describing a subset of the results in this paper appeared in [34].

Definitions and properties

We begin with a few definitions, following [35]. We think of a *string* x of *length* m as an array $x[0..m-1]$, where every $x[i]$, $0 \le i < m$, is a *letter* drawn from some fixed *alphabet* Σ of size $|\Sigma| = \mathcal{O}(1)$. We refer to any string $x \in \Sigma^q$ as a *q-gram*. The *empty string* of length 0 is denoted by ε. A string x is a *factor* of a string y if there exist two strings u and v, such that $y = uxv$. Let x be a non-empty string and y be a string. We say that there is an *occurrence* of x in y, or, simply, that x *occurs in* y, when x is a factor of y. The *Parikh vector* associated with a string $w \in \Sigma^*$ is denoted by $\mathcal{P}(w)$ and represents a vector of size $|\Sigma|$, where each component denotes the number of occurrences in w of the corresponding letter from Σ.

Consider the strings x, y, u, and v, such that $y = uxv$. If $u = \varepsilon$, then x is a *prefix* of y. If $v = \varepsilon$, then x is a *suffix* of y. We denote by $\mathtt{SA}$ the *suffix array* of y of length n, that is, an integer array of size n storing the starting positions of all lexicographically sorted suffixes of y, i.e. for all $1 \le r < n$, we have $y[\mathsf{SA}[r-1]..n-1] < y[\mathsf{SA}[r]..n-1]$ [33]. Let $\mathtt{lcp}(r, s)$ denote the length of the longest common prefix between $y[\mathsf{SA}[r]..n-1]$ and $y[\mathsf{SA}[s]..n-1]$, for all positions r, s on y, and 0 if they do not have a common prefix. We denote by $\mathtt{LCP}$ the *longest common prefix* array of y defined by $\mathtt{LCP}[r] = \mathsf{lcp}(r-1, r)$, for all $1 \le r < n$, and $\mathtt{LCP}[0] = 0$. The inverse $\mathtt{iSA}$ of the array $\mathtt{SA}$ is defined by $\mathsf{iSA}[\mathsf{SA}[r]] = r$, for all $0 \le r < n$. $\mathtt{SA}$, $\mathtt{iSA}$, and $\mathtt{LCP}$ of y can be computed in $\mathcal{O}(n)$ time and space [36].

A circular string of length m can be viewed as a traditional linear string which has the left- and right-most letters wrapped around and glued together in some way. Under this notion, the same circular string can be seen as m different linear strings, which would all be considered equivalent. Given a string x of length m, we denote by $x^i = x[i..m-1]x[0..i-1]$, $0 < i < m$, the ith *rotation* of x and $x^0 = x$. For instance, the string $x = x^0 = \mathtt{abababbc}$ has the following rotations: $x^1 = \mathtt{bababbca}$, $x^2 = \mathtt{ababbcab}$, and so on.

We give some further definitions following [28]. The *q-gram profile* of a string x is the vector $G_q(x)$, where $q > 0$ and $G_q(x)[v]$ denotes the total number of occurrences of q-gram $v \in \Sigma^q$ in x. The q-gram distance between two strings x and y is defined as

$$D_q(x,y) = \sum_{v \in \Sigma^q} \left| G_q(x)[v] - G_q(y)[v] \right|. \tag{1}$$

Note that D_q is a *pseudo-metric* as $D_q(x,y)$ can be 0 even if $x \neq y$. D_q has the following properties [28] for all $x, y, z \in \Sigma^*$ of length at least q.

1. Positivity: $D_q(x,y) \ge 0$
2. Symmetry: $D_q(x,y) = D_q(y,x)$
3. Triangular inequality: $D_q(x,y) \le D_q(x,z) + D_q(z,y)$
4. $|(|x| - |y|)| \le D_q(x,y) \le |x| + |y| - 2q - 2$
5. $D_q(x_1x_2, y_1y_2) \le D_q(x_1,y_1) + D_q(x_2,y_2) + 2(q-1)$, for $x_1, x_2, y_1, y_2 \in \Sigma^*$
6. $D_q(h(x), h(y)) \le D_q(x,y)$, for a non-length-increasing morphism h on Σ^*.

Example 1 Let $x = \mathtt{GGAGTCTA}$, $y = \mathtt{TTCTAGCG}$, and $q = 3$. Table 1 shows the q-gram profiles of strings x and y and the q-gram distance between them. Each row represents the frequency of a q-gram in the given string. For succinctness of presentation, only those rows with frequency greater than zero (in either string) are shown, as well as rows representing $\mathtt{AAA}$, $\mathtt{CCC}$, $\mathtt{GGG}$, and $\mathtt{TTT}$ as points of reference.

For a given integer parameter $\beta \ge 1$, we define a generalization of the q-gram distance in (1) by partitioning x and y in β *blocks* as evenly as possible, and computing the q-gram distance between each pair of blocks, one from x and one from y. The rationale is to enforce *locality* in the resulting overall distance. For the sake of presentation in the rest of the paper, we assume that the lengths $|x| = m$ and $|y| = n$ are both multiples of β, so that x and y are conceptually partitioned into β blocks, each of size m/β for x and n/β for y.

Definition 1 Given strings x of length m and y of length $n \ge m$ and integers $\beta \ge 1$ and $q > 0$, the *β-blockwise q-gram distance* $D_{\beta,q}(x,y)$ is defined as

$$D_{\beta,q}(x,y) = \sum_{j=0}^{\beta-1} D_q\left(x\left[\frac{jm}{\beta} .. \frac{(j+1)m}{\beta} - 1\right], y\left[\frac{jn}{\beta} .. \frac{(j+1)n}{\beta} - 1\right]\right). \tag{2}$$

Example 2 Following Example 1, let $x = \mathtt{GGAGTCTA}$ and $y = \mathtt{TTCTAGCG}$, $q = 3$, and $\beta = 2$. Further let $x_1 = \mathtt{GGAG}$, $x_2 = \mathtt{TCTA}$ and $y_1 = \mathtt{TTCT}$, $y_2 = \mathtt{AGCG}$ be

Table 1 q-gram profiles of strings x and y and q-gram distance $D_q(x, y) = 8$ between them

(a) $G_q(x)$	
AAA	0
AGC	0
AGT	1
CCC	0
CTA	1
GAG	1
GCG	0
GGA	1
GGG	0
GTC	1
TAG	0
TCT	1
TTC	0
TTT	0
(b) $G_q(y)$	
AAA	0
AGC	1
AGT	0
CCC	0
CTA	1
GAG	0
GCG	1
GGA	0
GGG	0
GTC	0
TAG	1
TCT	1
TTC	1
TTT	0
(c) $D_q(x, y)$	
AAA	0
AGC	1
AGT	1
CCC	0
CTA	0
GAG	1
GCG	1
GGA	1
GGG	0
GTC	1
TAG	1
TCT	0
TTC	1
TTT	0

the two blocks of x and y, respectively. Table 2 shows the q-gram profiles of strings x_1, x_2, y_1, and y_2; and the q-gram distance between x_1 and y_1 and the q-gram distance between x_2 and y_2.

Table 2 q-gram profiles of strings x_1, x_2, y_1, and y_2; q-gram distance between x_1 and y_1; and q-gram distance between x_2 and y_2, giving $D_{\beta,q}(x, y) = 8$

(a) $G_q(x_1)$	
AAA	0
AGC	0
AGT	0
CCC	0
CTA	0
GAG	1
GCG	0
GGA	1
GGG	0
GTC	0
TAG	0
TCT	0
TTC	0
TTT	0
(b) $G_q(y_1)$	
AAA	0
AGC	0
AGT	0
CCC	0
CTA	0
GAG	0
GCG	0
GGA	0
GGG	0
GTC	0
TAG	0
TCT	1
TTC	1
TTT	0
(c) $D_q(x_1, y_1)$	
AAA	0
AGC	0
AGT	0
CCC	0
CTA	0
GAG	1
GCG	0
GGA	1
GGG	0
GTC	0
TAG	0
TCT	1
TTC	1
TTT	0
(d) $G_q(x_2)$	
AAA	0
AGC	0
AGT	0

Table 2 continued

CCC	0
CTA	1
GAG	0
GCG	0
GGA	0
GGG	0
GTC	0
TAG	0
TCT	1
TTC	0
TTT	0
(e) $G_q(y_2)$	
AAA	0
AGC	1
AGT	0
CCC	0
CTA	0
GAG	0
GCG	1
GGA	0
GGG	0
GTC	0
TAG	0
TCT	0
TTC	0
TTT	0
(f) $D_q(x_2, y_2)$	
AAA	0
AGC	1
AGT	0
CCC	0
CTA	1
GAG	0
GCG	1
GGA	0
GGG	0
GTC	0
TAG	0
TCT	1
TTC	0
TTT	0

In this paper, we consider the following problem, where we search for the ith rotation of x that minimizes its blockwise distance from y as defined in (2). Ties are broken arbitrarily.

Circular Sequence Comparison (CSC)

Input: strings x and y of lengths m and $n \geq m$, respectively, and integers $\beta \geq 1$ and $q < m$

Output: i such that $D_{\beta,q}(x^i, y)$ is minimal

Algorithms

We use the following result to first give a naïve solution to the CSC problem.

Lemma 1 [28] *If we have space $\mathcal{O}(|\Sigma|^q)$ available, then the q-gram distance $D_q(x, y)$ can be computed in time $\mathcal{O}(m+n)$ and extra space $\mathcal{O}(m+n)$, where $m = |x|$ and $n = |y|$.*

We then apply Lemma 1 to each pair of blocks of x and y separately.

Lemma 2 *If we have space $\mathcal{O}(|\Sigma|^q)$ available, then the β-blockwise q-gram distance $D_{\beta,q}(x, y)$ can be computed in time $\mathcal{O}(m+n)$ and extra space $\mathcal{O}(\frac{m+n}{\beta})$, where $m = |x|$ and $n = |y|$.*

The naïve algorithm, denoted by `nCSC`, computes for $x' = xx$ the values

$$\delta_i = D_{\beta,q}(x'[i \,..\, i+m-1], y),$$

for all $0 \leq i < m$; we report position i such that δ_i is minimal. This requires the application of Lemma 2, m times. Therefore, we obtain the following.

Lemma 3 *If we have space $\mathcal{O}(|\Sigma|^q)$ available, then algorithm `nCSC` solves the CSC problem in time $\mathcal{O}(m(m+n))$ and extra space $\mathcal{O}(\frac{m+n}{\beta})$.*

Algorithm `hCSC`: a Heuristic algorithm

Here we give a simple heuristic algorithm, denoted by `hCSC`, to solve the CSC problem faster than `nCSC`, and return an approximation of the best rotation.

Step 1: We split $x' = xx$ in 2β non-overlapping string *blocks* of length m/β. We obtain strings $x_0, x_1, \ldots, x_{2\beta-1}$, such that $x_i = x'[\frac{im}{\beta} \,..\, \frac{(i+1)m}{\beta} - 1]$, for all $0 \leq i < 2\beta$. We split y in β non-overlapping string blocks of length n/β. We obtain strings $y_0, y_1, \ldots, y_{\beta-1}$, such that $y_i = y[\frac{in}{\beta} \,..\, \frac{(i+1)n}{\beta} - 1]$, for all $0 \leq i < \beta$.

Step 2: For a given sequence $x_j, \ldots, x_{j+\beta-1}$ of strings and y, we compute the β-blockwise q-gram distance as follows

$$\delta_j = D_{\beta,q}\left(x'\left[\frac{jm}{\beta} \,..\, \frac{jm}{\beta} + m - 1\right], y\right) = \sum_{i=0}^{\beta-1} D_q(x_{j+i}, y_i).$$

We compute δ_j, for all $0 \leq j \leq \beta$. We choose $j_{best} = j$ such that δ_j is minimal, for all $0 \leq j \leq \beta$. In other words, we have found a *window* of length m starting at position j_{best}, such that $(j_{best}+1) \bmod (m/\beta) = 0$, consisting of β blocks of length m/β each, that minimizes its β-blockwise q-gram distance from y.

Step 3: To perform a refinement on the position of the window, we consider all starting positions included in

the two blocks starting at positions j_{best} and $j_{best} - m/\beta$. This includes $2m/\beta - 1$ starting positions in total—we do not need to consider position $j_{best} - m/\beta$ as this was already considered by another window in Step 2. Similarly to Step 2, we obtain the β-blockwise q-gram distance δ_i between the window starting at position i and y, for all $j_{best} - m/\beta < i \leq j_{best} + m/\beta - 1$. We report position $i_{best} = i$ such that δ_i is minimal, for all $j_{best} - m/\beta < i \leq j_{best} + m/\beta - 1$.

Analysis Step 1 can be done trivially in time $\mathcal{O}(m+n)$. If we have space $\mathcal{O}(|\Sigma|^q)$ available, then, by Lemma 1, $D_q(x_{j+i}, y_i)$ can be computed in time $\mathcal{O}\left(\frac{m+n}{\beta}\right)$. By Lemma 2, δ_j can be computed in time $\mathcal{O}\left(\beta(\frac{m+n}{\beta})\right) = \mathcal{O}(m+n)$. Hence, Step 2 can be done in time $\mathcal{O}(\beta(m+n))$. In Step 3, the blockwise q-gram distance δ_i between a single window and y can be computed in time $\mathcal{O}\left(\beta(\frac{m+n}{\beta})\right) = \mathcal{O}(m+n)$. There exist $2m/\beta - 1$ such windows. Hence, Step 3 can be done in time $\mathcal{O}\left(\frac{m(m+n)}{\beta}\right)$. Overall, the algorithm requires time $\mathcal{O}\left(\beta(m+n) + \frac{m(m+n)}{\beta}\right)$ and space $\mathcal{O}(|\Sigma|^q + m + n)$.

For practical purposes, setting $\beta = \mathcal{O}(\sqrt{m})$ and $q = \mathcal{O}(\log_{|\Sigma|} m)$ gives an algorithm with time complexity $\mathcal{O}(\sqrt{m}(m+n))$ and space complexity $\mathcal{O}(m+n)$.

Algorithm `saCSC`: an exact suffix-array-based algorithm

The above heuristic `hCSC` does not guarantee to find the exact value i, for which $\delta_i = D_{\beta,q}(x^i, y)$ is minimal. In particular, when we identify j_{best} in Step 2, that is, the j for which δ_j is minimal, we take into account only the values of j such that $(j+1) \bmod (m/\beta) = 0$. Thus, Step 3 cannot guarantee that i_{best}, the local minimum obtained by shifting the window m/β positions to the right and left of j_{best}, is minimal for all $0 \leq i < m$. In this section, we give a fast and exact algorithm, denoted by `saCSC`, to find i such that $\delta_i = D_{\beta,q}(x^i, y)$ is minimal, based on the suffix array (see "Definitions and properties" section).

We partially follow the idea from [37]. This work investigates the string matching problem in the setting of k-abelian equivalences: two strings are considered k-abelian equivalent for some positive integer k, if they have the same length and share the same factors of length at most k, including multiplicities. Note that if k is greater than or equal to the string's length, then the strings must be equal. A version of this result, called extended k-abelian equivalence, focuses only on the factors of length k. By setting $k = q$, it is quite straightforward to notice the equivalence with q-grams. Therefore, in order to avoid confusion we will refer to the former notion from now on as *q-abelian equivalence*.

In [37], the authors propose a linear-time algorithm to solve the string matching problem when looking at q-abelian equivalent strings: given a string x of length m, a string y of length $n \geq m$, and a positive integer $q < m$, all factors of y that are q-abelian equivalent to x can be found in time and space $\mathcal{O}(m+n)$. The idea of the algorithm in [37] consists in constructing the suffix array of the string xy, and ranking sets of identical q-length prefixes of suffixes in the suffix array in the order of their appearance. Then it constructs new strings based on this ranking, and solves the problem as in the *jumbled matching* case [38], i.e. identifying all factors of y that have the same Parikh vector as x.

We first describe our algorithm for a single block ($\beta = 1$) and then address the general case ($\beta \geq 1$).

Basic algorithm for $\beta = 1$. We construct the suffix array of the string xxy and assign a *rank* to the prefix with length q of each suffix with length at least q, based on its order in the suffix array. That is, the first i_0 suffixes, of length at least q, in the suffix array, all sharing the same prefix of length q, will get rank 0; the next i_1 suffixes, of length at least q, sharing the same prefix of length q, different from the previous one, will get rank 1, and so on. Next, based on this ranking, we construct two new strings x' of length $2m - q + 1$ and y' of length $n - q + 1$, such that $x'[i] = j$, if j is the rank of the q-length prefix of the $(i+1)$th suffix of xx in the suffix array of xxy (the same goes for y). It is not difficult to see that the ranks go up at most to value $m + n - q + 1$. However, we can reduce this value to $m + 2$ by introducing two new ranks a_x and a_y: we can conceptually replace by a_x every letter of x' that does not occur in y', and by a_y every letter of y' that does not occur in x'. Hence we can consider that the new strings x' and y' are defined over an integer alphabet of size *at most* $\min(n - q + 1, m) + 2 \leq m + 2$.

Example 3 Let $x =$ `GAGTCTA`, $y =$ `TCTAGCG`, and $q = 3$. We denote xxy by z.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$z[i]$	G	A	G	T	C	T	A	G	A	G	T	C	T	A	T	C	T	A	G	C	G
SA[i]	6	17	1	8	13	19	4	15	11	20	0	7	18	2	9	5	16	12	3	14	10
LCP[i]	0	2	2	6	1	0	1	4	3	0	1	7	1	1	5	0	3	2	1	5	4
$x'[i]$	a_x	a_x	a_x	2	0	1	a_x	a_x	a_x	a_x	2	0									
$y'[i]$	2	0	1	a_y	a_y																

Here, $x'[3] = y'[0] = 2$ denotes that $x[3..5] = y[0..2] = \texttt{TCT}$ and $x'[0] = a_x$ denotes that $x[0..2] = \texttt{GAG}$ does not occur in y.

We observe that when identifying the q-gram distance between two blocks, we can apply the idea in [37], with the only difference that we should also maintain a Parikh vector that stores the *differences* between the number of occurrences of q-grams (in fact the new letters given by the ranks) in the current block of xx and y. Moreover, at the time of the construction of y', we also construct a Parikh vector $\mathcal{P}(y')$, storing for each letter of y', the number of its occurrences in y'. Notice that $|\mathcal{P}(y')| \leq m + 2$. Later on, when computing the q-gram distances, we can construct another vector diff to store the letter differences between $\mathcal{P}(y')$ and the Parikh vector covering the $m - q + 1$ letters of x' associated with a window of length m on the string xx. This gives us the current Parikh difference and, in fact, represents the q-gram distance between the two analyzed blocks, where $|\mathsf{diff}| \leq m + 2$. Apart from these, we only need another vector δ of size m, which stores at each position i the actual q-gram distance δ_i between y and the window starting at position i in xx, which is the ith rotation x^i of x.

We use a sliding window of length m to maintain the above information. When the window is shifted one position to the right, we have to add to the difference-vector diff the previous first element of the window, and deduct from it the current last element of it. The distance δ_i between y' and the factor of x' starting at position i is thus updated using, in addition, the value of the q-gram distance δ_{i-1} as follows. If, after adding the previous first element to the vector, we have a non-positive value at this position, we update the distance by decreasing the previous value by 1; otherwise, we increase it by 1. If, after deducting the current last element to the vector, we have a non-negative value at this position, we update the distance by decreasing the previous value by 1; otherwise, we increase it by 1. The distance will never be less than the number of occurrences of a_y. Furthermore, if the previous first element was a_x, the new distance decreases by 1, and for every newly added a_x, it increases by 1. As these operations require constant time, after going once through x' with y', we obtain the list of distances δ_i from y to each rotation x^i in linear time.

We are now able to give a more formal description of the steps to solve the CSC problem for $\beta = 1$, which follow a dynamic programming scheme.

Step 1: Construct the SA, iSA, and LCP of xxy. Rank the q-length prefixes of suffixes using LCP-array queries. Construct x' and y', as well as $\mathcal{P}(y')$, the Parikh vector storing, for each letter of y', the number of its occurrences in y'; making proper use of letters a_x and a_y, the ranks that do not occur in either y' or x', respectively. Further, create $\mathsf{diff} = \mathcal{P}(y')$ and $\delta_0 = \sum_{i=0}^{|\mathcal{P}(y')|-1} \mathcal{P}(y')[i]$.

Example 4 Following Example 3, let $x = \texttt{GAGTCTA}$, $y = \texttt{TCTAGCG}$, $q = 3$, and $z = xxy$.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$z[i]$	G	A	G	T	C	T	A	G	A	G	T	C	T	A	T	C	T	A	G	C	G
$x'[i]$	a_x	a_x	a_x	2	0	1	a_x	a_x	a_x	a_x	2	0									
$y'[i]$	2	0	1	a_y	a_y																

The table below represents vector diff, right after the execution of Step 1, which implies that $\delta_0 = 5$.

a_x	0
a_y	2
0	1
1	1
2	1

Step 2: Read the first $m - q + 1$ letters of x', which constitute our sliding window of length m on the string xx. When reading letter $x'[i]$, update diff by decreasing by 1 the value of the newly read letter, and update δ_0, by either increasing the current value of the distance when there were read too many of the current letters, or decreasing it, when more of these letters still occur in y'

$$\mathsf{diff}[x'[i]] = \mathsf{diff}[x'[i]] - 1 \text{ and}$$
$$\delta_0 = \begin{cases} \delta_0 - 1, & \text{if } \mathsf{diff}[x'[i]] \geq 0 \\ \delta_0 + 1, & \text{if } \mathsf{diff}[x'[i]] < 0. \end{cases}$$

Example 5 Following Example 4, the table below represents vector diff, right after the execution of Step 2, which implies that $\delta_0 = 6$.

a_x	3
a_y	2
0	0
1	1
2	0

Step 3: Let i be the current position in x' and repeat this step, one position at a time. Shift the window to the right, update the information for diff

$$\mathsf{diff}[x'[i]] = \mathsf{diff}[x'[i]] + 1 \text{ and}$$
$$\mathsf{diff}[x'[i+m]] = \mathsf{diff}[x'[i+m]] - 1,$$

and calculate δ_{i+1}, based on this information, sequentially applying the two following rules

$$\delta_{i+1} = \begin{cases} \delta_i - 1, & \text{if } \mathsf{diff}[x'[i]] \leq 0 \\ \delta_i + 1, & \text{if } \mathsf{diff}[x'[i]] > 0 \end{cases}$$

$$\delta_{i+1} = \begin{cases} \delta_{i+1} - 1, & \text{if } \mathsf{diff}[x'[i+m]] \geq 0 \\ \delta_{i+1} + 1, & \text{if } \mathsf{diff}[x'[i+m]] < 0. \end{cases}$$

Example 6 Following Example 5, the table below represents vector diff at iteration $i' = 3$ of Step 3, which implies that $\delta_0 = 4$. This is in fact the best rotation of x, that is, $x^3 =$ TCTAGAG.

a_x	2
a_y	2
0	0
1	0
2	0

Correctness Steps 1 and 2 are trivially correct as at the end of them we have that diff is the difference between $\mathcal{P}(y')$ and the vector corresponding to the window. These operations follow directly from the definitions of SA and LCP, and are followed by a simple traversal of the suffix array in order to obtain the ranks and create the $\mathcal{P}(y')$ and diff vectors. Also, δ_0, which was initially the number of letters in y', is decreasing as long as the difference between the vectors for a specific letter is non-negative (thus, we still have more occurrences of that letter in y' compared to the window), and increasing otherwise. In Step 3, we update the difference vector by increasing the value at position $x'[i]$ and decreasing that of the new letter $x'[i+m]$ added to the difference. The q-gram distance at that position is based on the values of the newly obtained difference vector, as well as the q-gram distance at the previous position: if $\mathsf{diff}[x'[i]] \leq 0$, then obviously there were more letters $x'[i]$ in y' than in the window, thus we need to decrease, while, if $\mathsf{diff}[x'[i]] > 0$, then there were at least as many letters $x'[i]$ in the window as in y', and taking one out increases the distance. The complementary reasoning applies to the newly added letter $x'[i+m]$. The value of δ_i never goes below the number of occurrences of a_y in y' (it is equal to that, when all other elements of diff are 0) and represents the q-gram distance between y and x^i, the corresponding window of length m starting at position i in xx.

Analysis In Step 1, constructing SA, iSA, and LCP of xxy can be done in time and extra space $\mathcal{O}(m+n)$ ("Definitions and properties" section). Furthermore, the construction of x', y', $\mathcal{P}(y')$, diff, and δ_0 is done with the same time and space cost. In Step 2, updating diff and δ_0 after reading each letter takes constant time, as we execute two operations, thus $\mathcal{O}(m)$ in total. Constant time is required for each iteration in Step 3 to compute the value of δ_i, $1 \leq i < m$, and update diff, since a constant number of operations are executed, thus $\mathcal{O}(m)$ in total. Hence, we can solve the CSC problem for $\beta = 1$ in time and space $\mathcal{O}(m+n)$.

General algorithm for $\beta \geq 1$. We can now generalize this algorithm to solve the CSC problem for any $\beta \geq 1$, which gives algorithm saCSC. We maintain a Parikh vector for each block, and apply the above basic algorithm for the jth block in each string, computing their q-gram distance. If we denote by $\mathcal{P}_j(y')$ and diff_j, for all $0 \leq j < \beta$, the β Parikh vectors of y' and of the q-gram distances, respectively, as well as by $\delta_{i,j}$ the q-gram distance between the jth block of y and x^i, then the updates will be given by the formulae below. Hence, at each position $i < m$, we can update all of the β Parikh vectors corresponding to the blocks, as previously described, in time $\mathcal{O}(\beta)$. As an example, see here the modification of the previous Step 3, with the other two steps being easily adapted in a similar fashion.

Step 3': When shifting the window one position to the right from position i, update the information for every diff_j, where $0 \leq j < \beta$, as follows

$$\mathsf{diff}_j\left[x'\left[i + \frac{jm}{\beta}\right]\right] = \mathsf{diff}_j\left[x'\left[i + \frac{jm}{\beta}\right]\right] + 1$$

$$\mathsf{diff}_j\left[x'\left[i + \frac{(j+1)m}{\beta}\right]\right] = \mathsf{diff}_j\left[x'\left[i + \frac{(j+1)m}{\beta}\right]\right] - 1,$$

and calculate $\delta_{i+1,j}$, based on this information, sequentially applying the two following rules

$$\delta_{i+1,j} = \begin{cases} \delta_{i,j} - 1, & \text{if } \mathsf{diff}_j\left[x'[i + \frac{jm}{\beta}]\right] \leq 0 \\ \delta_{i,j} + 1, & \text{if } \mathsf{diff}_j\left[x'[i + \frac{jm}{\beta}]\right] > 0 \end{cases}$$

$$\delta_{i+1,j} = \begin{cases} \delta_{i+1,j} - 1, & \text{if } \mathsf{diff}_j\left[x'[i + \frac{(j+1)m}{\beta}]\right] \geq 0 \\ \delta_{i+1,j} + 1, & \text{if } \mathsf{diff}_j\left[x'[i + \frac{(j+1)m}{\beta}]\right] < 0. \end{cases}$$

Therefore, we obtain the following result.

Theorem 1 *Algorithm* saCSC *solves the* CSC *problem in* $\mathcal{O}(\beta m + n)$ *time and space.*

Implementation

We implemented algorithms nCSC, hCSC, and saCSC as the program CSC. Given one of the three methods, two sequences x and y in (Multi)FASTA format, the number β of blocks, and the length q of the q-grams, CSC finds the rotation of x (or an approximation of it) that minimizes its β-blockwise q-gram distance from y. The implementation is distributed under the GNU General Public License (GPL), and it is available freely at http://www.github.com/solonas13/csc.

For comparison purposes, we implemented a naïve algorithm that compares all rotations of x against y using the Needleman-Wunsch algorithm [39] with substitution matrices and affine gap penalty scores [40]; we denote this implementation by `cNW`. We also implemented the following heuristics. We first use the Smith-Waterman local alignment algorithm [41] to search for the best local alignment of x and y and then use a central match from this local alignment to anchor the global alignment (see also [11]); we denote this implementation by `hSW`.

Refining algorithm `saCSC`

The application of the β-blockwise q-gram distance via algorithm `saCSC` suggests that an optimal or a close-to-optimal rotation of x can be found when compared to `cNW`. Due to the locality property offered by the newly introduced distance notion, it is reasonable to assume that the close-to-optimal rotation returned by `saCSC` may be refined via some quick heuristics that take into consideration the blocks at both ends.

Let x^i be the close-to-optimal rotation of x returned by `saCSC`. We introduce a new input parameter $0 < p \leq \frac{\beta}{3}$, which defines the length L of the prefixes and suffixes of x^i and y to be considered in the refinement as follows:

$$L = \left\lfloor p \times \frac{m}{\beta} \right\rfloor.$$

We take p block(s) of the *prefix* of x^i, concatenate it with a string of equal length L comprised only of letter \$, where $\$ \notin \Sigma$, and concatenate that with p block(s) of the *suffix* of x^i to form a new string x'' of length $3L$. We do the same with y to form a new string y''.

The refinement algorithm works by taking all rotations of x'' and comparing their similarity to y''. Each rotation of x'' is compared to y'' excluding when a \$ letter is found at index 0 of the rotation of x''. We measure the similarity between the strings for which equality between letters are positively valued; inequalities, insertions, and deletions are negatively valued; and comparisons involving \$ are neither positively nor negatively valued. The goal of rotating x'' serves to find the rotation that maximizes the similarity to y'' and, to this end, we make use of the Needleman-Wunsch algorithm. The rotation of x'' which results in the maximum score is chosen as the best rotation, and hence, the final rotation $x^{i'}$ of x is computed based on this rotation of x''. Ties are broken arbitrarily. We denote this new algorithm, consisting of `saCSC` and the refinement stage, by `saCSCr`.

The application of the Needleman-Wunsch algorithm on strings of length $3L$ has a time complexity of $\mathcal{O}(L^2)$. Considering all rotations of x'' results in a time complexity of $\mathcal{O}(L^3)$ for the refinement step. Overall, `saCSCr` takes time $\mathcal{O}(\beta m + n + L^3)$.

Example 7 Consider the following pair of strings obtained from `saCSC`

x^i = GACACCCCCCACAGTTTATGTAGCTT...ACCCCGAACCAACCAAACCCCAAA
y = GTTTATGTAGCTTACCTCCCCAAAGC...CAAACCCCAAAGACACCCCACACA

and the following pair of strings formed for $L = 25$.

x'' = GACACCCCCCACAGTTTATGTAGCTT$$$$$$$$$$$$$$$$$$$$$$$$$ACCCCGAACCAACCAAACCCCAAA
y'' = GTTTATGTAGCTTACCTCCCCAAAG$$$$$$$$$$$$$$$$$$$$$$$$$CAAACCCCAAAGACACCCCACACA

The Needleman-Wunsch algorithm for all rotations of x'' and string y'' gives the following optimal alignment

GTTTATGTAGCTT$$$$$$$$$$$$$$$$$$$$$$$$$ACCCCGAACCAACCAAACCCCAAAGACACCCCCCACA
GTTTATGTAGCTTACCTCCCCAAAG$$$$$$$$$$$$$$$$$$$$$$$$$CAAACCCCAAAGACACCCCACACA

which tells us that a refined rotation is in fact $x^{i'}$, where $i' = i - 13$

$x^{i'}$ = GTTTATGTAGCTT...CAAACCCCAAAGACACCCCCCACA
y = GTTTATGTAGCTT...CAAACCCCAAAGACACCCCACACA

Experimental results

The following experiments were conducted on a desktop computer using one core of Intel® Core™ i7-2600 CPU at 3.4GHz and 12GB of RAM under 64-bit GNU/Linux. All programs were compiled with `gcc` version 4.7.3. We used both synthetic data and real data. All input datasets referred to in this section are publicly maintained at the same web-site. First, in "Accuracy"–"Time performance" sections, we establish the quality (accuracy and performance) of our methods. Then, in "Application to synthetic data"–"Application to real data" sections, we show applications of our methods.

Accuracy

We began with simulating three DNA sequence datasets using INDELible [42], with each dataset consisting of 12 sequences (denoted by α), each of length approximately 2500 bp (denoted by γ). INDELible produces linear sequences with substitutions, insertions, and deletions at rates defined by the user. Three unique substitution rates (denoted by θ) were set, per dataset, using the substitution model JC69 (Jukes-Cantor, 69): 5, 20, and 35 %. The insertion and deletion rates were set, respectively, to 4 and 6 % (denoted by κ and ω), relative to substitution rate of one, similar to those observed in MtDNA in primates and mammals [25]. We refer to these datasets as *Original*.

To allow for comparison of the performance of the algorithms in realigning randomly rotated sequences, which should be similar to those obtained from sequencing circular DNA structures, such as MtDNA, one random rotation was generated in each sequence in all datasets, creating new datasets which will be referred to as *Random*. Using the three *Random* datasets allowed us to test the accuracy of `hCSC` and `saCSC`; notice that `nCSC` and `saCSC` always return the same rotation. For each *Random* dataset, an all-against-all sequence comparison was performed. That is, all 66 possible pairs of sequences in each dataset were given as input to both `hCSC` and `saCSC`. β was set to $\lceil\sqrt{m}\rceil = 50$ and q was set to $\lceil\log_{|\Sigma|} m\rceil = 6$. The resultant re-rotated sequences were aligned using EMBOSS Needle (default parameters) and the similarity scores were compared to those of the *Original* and *Random* datasets, which were input directly to EMBOSS Needle (default parameters). The results can be found in Fig. 1.

The results show that: (a) `hCSC` and `saCSC` yield significantly improved similarity scores compared to those obtained from giving *Random* datasets as input directly to EMBOSS Needle; and (b) `hCSC` and `saCSC` yield similarity scores that are identical or almost identical—notice that the black (Original), green (`hCSC`), and blue (`nCSC`/`saCSC`) points *coincide*—to those obtained from giving *Original* datasets as input directly to EMBOSS Needle. This implies that algorithms `hCSC`, `nCSC`, and `saCSC` return the rotation maximizing the similarity score for all pairwise comparisons.

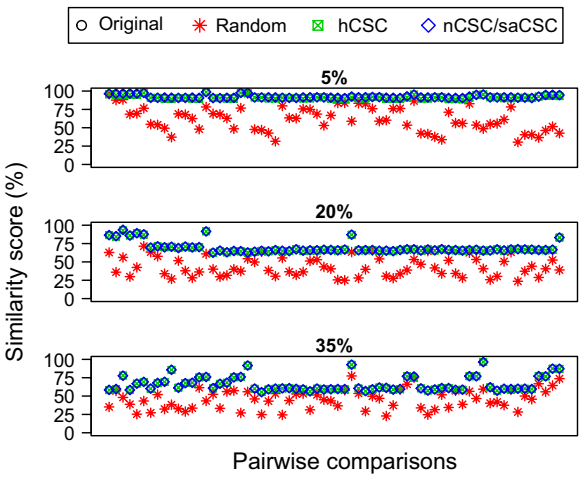

Fig. 1 Accuracy. Accuracy comparison for substitution rates 5, 20, and 35 %; the *black*, *green*, and *blue points* coincide implying that algorithms `hCSC`, `nCSC`, and `saCSC` return the rotation maximizing the similarity score for all pairwise comparisons

Hence, we establish here that the introduced distance measure coupled with the respective algorithms consistently yield a very high accuracy, compared to the standard measure [17, 39, 40], for both *low* and *high* substitution rates.

Time performance

We then compared the time performance of the algorithms. Each algorithm was given a pair of randomly generated sequences starting from $m = n = 50$ bp and doubling 8 times to a length of $m = n = 12,800$ bp. It was expected that the slowest algorithm would be `cNW` which runs in time $\mathcal{O}(nm^2)$. Then it would be algorithm `nCSC` which runs in time $\mathcal{O}(m(m+n))$, then algorithm `hCSC`, which runs in time $\mathcal{O}\left(\beta(m+n) + \frac{m(m+n)}{\beta}\right)$, and lastly algorithm `saCSC`, which runs in time $\mathcal{O}(\beta m + n)$.

Initially, β was set to $\lceil\sqrt{m}\rceil$ and q was set to $\lceil\log_{|\Sigma|} m\rceil$. The results in Fig. 2 demonstrate orders-of-magnitude superiority of `saCSC` compared to `cNW` and `nCSC`, confirming our theoretical findings. Algorithm `hCSC` is the second fastest. Although β was set to $\lceil\sqrt{m}\rceil$, `saCSC` clearly outperforms `hCSC`, due to the use of a highly optimized implementation of the suffix-array construction [43], thus highlighting the importance of suitably implemented data structures such as suffix arrays.

Since the time complexities of `hCSC` and `saCSC` depend on β, we repeated the same experiment with these two algorithms setting β to $\lceil m/25\rceil$ and q to $\lceil\log_{|\Sigma|} m\rceil$—

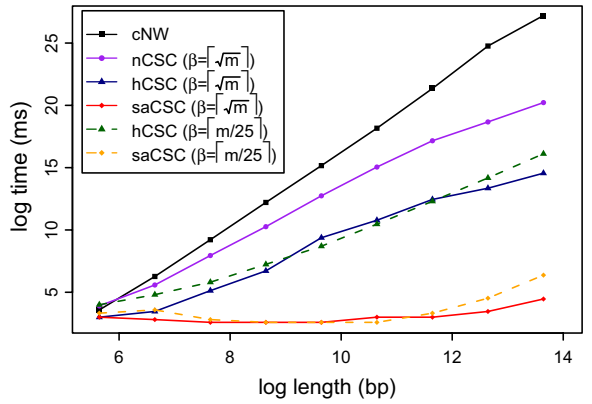

Fig. 2 Time performance. Elapsed-time comparison of algorithms `cNW`, `nCSC`, `hCSC`, and `saCSC`

notice that q does not affect the time efficiency of the algorithms. The results in Fig. 2 show that `hCSC` and `saCSC` are still the fastest, even though $m = \mathcal{O}(\beta)$, and that `saCSC` is clearly the fastest of all. As expected for $m = \mathcal{O}(\beta)$, we observe that `hCSC` and `saCSC` become gradually slower as m grows.

More algorithms could have been included in the comparison but their (at least) quadratic time complexity [22, 23] prevents them from competing with `saCSC`.

Application to synthetic data

For evaluating the proposed methods for circular sequence comparison in some relevant application, we also implemented the following pipeline for distance-based phylogenetic reconstruction of a dataset with N circular sequences.

1. For each pair (x, y) of the N sequences, we use one method for circular sequence comparison to compute the best rotation x^i.
2. A *similarity* score for (x^i, y) is then computed using EMBOSS Needle (default parameters) and stored in cell $[x, y]$ of an $N \times N$ similarity score matrix.
3. The similarity score matrix is transformed into a distance matrix by converting each score into a *distance* relative to the maximum score in the similarity score matrix.
4. Neighbour joining clustering is performed on the distance matrix, using NINJA [44], to produce a phylogenetic tree.

Phylogenetic trees were constructed by NINJA [44], for the aforementioned *Random* datasets, using output from the following algorithms: `cNW` (EMBOSS default parameters), `hSW` (see introduction of "Implementation" section, EMBOSS default parameters), and `saCSCr` ($\beta = 50$, $q = 5$, and $p = 1$). Notice that, output from `cNW` should be the same as from EMBOSS Needle (default parameters) with the *Original* datasets as input. In terms of accuracy, the Robinson-Foulds (RF) distance metric [45, 46] was used to compare the three resultant phylogenetic trees with the tree resulting from EMBOSS Needle (default parameters) on the *Original* and *Random* datasets, denoted by `NW(o)` and `NW(r)`, respectively. RF distance can be defined as the number of operations required to transform one tree in to another. If two isomorphic trees share the same labelling then they have an RF distance of 0. The results displayed in Table 3 clearly show that `saCSCr` and `cNW` produce the most accurate results with these nine datasets. As also shown in [11], `hSW` followed by EMBOSS Needle (default parameters) can often result in sub-optimal global alignments.

In terms of time performance, the elapsed time required for each method to process each dataset was recorded and the results are displayed in Table 4. It is clear, from the results presented heretofore, that `saCSCr` outperforms all other algorithms by at least one order of magnitude.

Application to real data

We have concluded thus far that using $\beta = \lceil\sqrt{m}\rceil$ and $q = \lceil\log_{|\Sigma|} m\rceil$ results in a reasonable trade-off between running time and accuracy. In the following section, where necessary, we adopt these values and multiply or divide them by a constant factor (factor of two), depending on the length of the input sequences.

DNA sequences

Pairwise sequence comparison. As the input dataset, we used two real sequences from GenBank: human

Table 3 RF distances between the tree obtained from the `NW(o)` and those obtained from `NW(r)`, `cNW`, `hSW`, and `saCSCr`

Dataset $<\alpha, \gamma, \theta, \kappa, \omega>$	`NW(r)`	`cNW`	`hSW`	`saCSCr`
<12, 2500, 0.05, 0.06, 0.04>	16	0	0	0
<12, 2500, 0.20, 0.06, 0.04>	12	0	0	0
<12, 2500, 0.35, 0.06, 0.04>	4	0	0	0
<25, 2500, 0.05, 0.06, 0.04>	44	0	0	0
<25, 2500, 0.20, 0.06, 0.04>	24	0	0	0
<25, 2500, 0.35, 0.06, 0.04>	16	0	0	0
<50, 2500, 0.05, 0.06, 0.04>	86	0	6	0
<50, 2500, 0.20, 0.06, 0.04>	84	0	0	0
<50, 2500, 0.35, 0.06, 0.04>	56	0	0	0

The number of sequences in the dataset is denoted by α; γ denotes their lengths; θ denotes the substitution rate; κ and ω denote the relative insertion and deletion rates, respectively

Table 4 Elapsed-time comparison (in seconds) for algorithms cNW, hSW, and saCSCr

Dataset $< \alpha, \gamma, \theta, \kappa, \omega >$	cNW	hSW	saCSCr
<12, 2500, 0.05, 0.06, 0.04>	10,139.36	72.43	6.90
<12, 2500, 0.20, 0.06, 0.04>	9888.84	80.91	6.57
<12, 2500, 0.35, 0.06, 0.04>	10,052.33	80.16	6.28
<25, 2500, 0.05, 0.06, 0.04>	46,311.85	369.02	27.61
<25, 2500, 0.20, 0.06, 0.04>	46,230.07	375.41	28.92
<25, 2500, 0.35, 0.06, 0.04>	46,289.99	400.30	30.44
<50, 2500, 0.05, 0.06, 0.04>	122,165.95	1563.96	125.63
<50, 2500, 0.20, 0.06, 0.04>	121,810.69	1617.89	123.12
<50, 2500, 0.35, 0.06, 0.04>	120,679.32	1662.82	123.77

The number of sequences in the dataset is denoted by α; γ denotes their lengths; θ denotes the substitution rate; κ and ω denote the relative insertion and deletion rates, respectively

(NC_001807) and chimpanzee (NC_001643) MtDNA sequences. The MtDNA genome size for human is 16,571 bp and for chimpanzee is 16,554 bp. Their pairwise sequence alignment using EMBOSS Needle (default parameters) gives a similarity of 85.1 %. We used cNW (EMBOSS default parameters) to obtain the rotation of NC_001807 that maximizes its similarity score with NC_001643. This experiment took approximately 28 h and the resultant rotation 578 of NC_001807 improved the similarity score to 91 %. This result was then compared to those obtained from saCSC (equivalent to saCSCr with $p = 0$) and saCSCr with varying parameters, displayed in Table 5.

The convergence of the results after the additional step of refinement (see Table 5 in italics) demonstrates the convenience and necessity of saCSCr.

For clarity of presentation hereafter, instead of using β, we denote by ℓ the length of the block chosen in algorithm saCSCr.

Table 5 Rotations of GenBank sequence NC_001807 obtained when compared to NC_001643 with varying parameters of saCSCr

q	β	p	Rotation
5	50	0	566
5	50	1	*578*
5	$\sqrt{m}$	0	567
5	$\sqrt{m}$	1	*578*
5	$2\sqrt{m}$	0	583
5	$2\sqrt{m}$	1	*578*
5	$\frac{\sqrt{m}}{2}$	0	566
5	$\frac{\sqrt{m}}{2}$	1	*578*

We repeated this experiment with the human and gorilla (NC_011120) MtDNA sequences. The MtDNA genome size for gorilla is 16,412 bp. Their pairwise sequence alignment using EMBOSS Needle (default parameters) gives a similarity of 83.5 %. After using saCSCr to rotate sequence NC_001807 ($\ell = 50$, $q = 5$, and $p = 1$), EMBOSS Needle (default parameters) gave a significantly improved similarity of 88.4 %.

Finally, note that the experiments which used saCSC and saCSCr each took a fraction of a second to run.

Distance-based phylogenetic reconstruction Three datasets of 16 primate, 12 mammalian and 19 mixed mammalian and primate MtDNA sequences, of average length 16,500 bp, were obtained from GenBank. We followed the same pipeline as described in "Application to synthetic data" section. The RF distance between the trees produced by cNW (EMBOSS default parameters), and the trees produced by saCSCr ($\ell = \lceil\sqrt{m}\rceil = 129$, $q = 5$, and $p = 1$) followed by EMBOSS Needle (default parameters), was 0.

RNA sequences

Eighteen viroid sequences were obtained from RefSeq, a database of curated molecular biological sequences [47]. Their lengths and target hosts vary, ranging from 348 to 371 bp and infecting peppers and citrus fruits, respectively. We followed the same pipeline as described in "Application to synthetic data" section. The RF distance between the tree produced by cNW (EMBOSS default parameters), and the tree produced by saCSCr ($\ell = \lceil\sqrt{m}\rceil = 19$, $q = \lceil\log_{|\Sigma|} m\rceil = 5$, and $p = 1$) followed by EMBOSS Needle (default parameters), was 0.

Protein sequences

Linear, circularly-permuted protein sequences Eight sequences of proteins, of average length 950 amino acids, belonging to β-glucosidase family [48] were obtained from the UniProt protein database [49]. We followed the same pipeline as described in "Application to synthetic data" section. The RF distance between the tree produced by cNW (EMBOSS default parameters), and the tree produced by saCSCr ($\ell = \lceil\sqrt{m}\rceil = 31, q = \lceil\log_{|\Sigma|} m\rceil = 5$, and $p = 1$) followed by EMBOSS Needle (default parameters), was 0.

Naturally-occurring circular proteins Ten bacteriocin protein sequences, of average length 20 amino acids, were obtained from Cybase [50], a database of cyclical protein sequences. We followed the same pipeline as described in "Application to synthetic data" section. The RF distance between the tree produced by cNW (EMBOSS default parameters), and the tree produced by saCSCr ($\ell = 2\lceil\sqrt{m}\rceil = 10$, $q = 2\lceil\log_{|\Sigma|} m\rceil = 6$, and $p = 1$) followed by EMBOSS Needle (default parameters), was 0.

Conclusions

In this paper, we introduced a new distance measure for sequence comparison based on q-grams, and showed how it can be applied *effectively* and computed *efficiently* for circular sequence comparison. The most efficient algorithm presented here, `saCSC`, solves our defined problem CSC, exactly. Extensive experimental results, using both real and synthetic data, show that it maintains an accuracy very competitive to the optimal obtained after considering all rotations of x against y naïvely using global alignments. We also showed that algorithm `saCSCr` can bridge the gap between the optimal solution and our approximation via an additional refinement step. Finally, the presented experimental study demonstrates orders-of-magnitude superiority of our approach in terms of runtime efficiency. Our immediate target is to implement algorithm `saCSCr` in `BEAR` [24], a state-of-the-art tool for improving multiple circular sequence alignment.

Authors' contributions

RG, CSI, RM, NP, and SPP devised the algorithms. SPP, AR, and FV implemented the algorithms. AR and FV conducted the experiments. All authors contributed equally in writing up the manuscript. All authors read and approved the final manuscript.

Author details

[1] Department of Informatics, University of Pisa, Pisa, Italy. [2] ERABLE team, INRA, Paris, France. [3] Department of Informatics, King's College London, London, UK. [4] Department of Computer Science, Kiel University, Kiel, Germany.

Acknowledgements

The publication costs for this article were funded by the Open Access funding scheme of King's College London. RG and NP have been partially supported by the Italian Ministry of Education, Universities, and Research (MIUR) under PRIN 2012C4E3KT national research project AMANDA—Algorithmics for Massive and Networked Data, and by the University of Pisa under PRA 2015 project Computational Methods for Personalized Medicine. RM is supported by the P.R.I.M.E. programme of DAAD co-funded by BMBF and the EU's 7th Framework Programme (#605728), and the Newton International Fellowship co-funded by the Royal Society and the British Academy. FV is supported by an EPSRC Grant (Doctoral Training Grant #EP/M506357/1).

Competing interests

The authors declare that they have no competing interests.

Received: 8 December 2015 Accepted: 25 April 2016
Published online: 10 May 2016

References

1. Craik DJ, Allewell NM. Thematic minireview series on circular proteins. J Biol Chem. 2012;287(32):26999–7000.
2. Helinski DR, Clewell DB. Circular DNA. Annu Rev Biochem. 1971;40:899–942.
3. Del Castillo CS, Hikima JI, Jang HB, Nho SW, Jung TS, Wongtavatchai J, Kondo H, Hirono I, Takeyama H, Aoki T. Comparative sequence analysis of a multidrug-resistant plasmid from Aeromonas hydrophila. Antimicrob Agents Chemother. 2013;57:120–9.
4. Taanman JW. The mitochondrial genome: structure, transcription, translation and replication. Biochem Biophys Acta Bioenerg. 1999;1410(2):103–23.
5. Goios A, Pereira L, Bogue M, Macaulay V, Amorim A. mtDNA phylogeny and evolution of laboratory mouse strains. Genome Res. 2007;17(3):293–8.
6. Wang Z, Wu M. Phylogenomic reconstruction indicates mitochondrial ancestor was an energy parasite. PLoS One. 2014;10(9):e110685.
7. Cohen S, Houben A, Segal D. Extrachromosomal circular DNA derived from tandemly repeated genomic sequences in plants. Plant J. 2008;53(6):1027–34.
8. Kuttler F, Mai S. Formation of non-random extrachromosomal elements during development, differentiation and oncogenesis. Semin Cancer Biol. 2007;17:56–64.
9. Brodie R, Smith AJ, Roper RL, Tcherepanov V, Upton C. Base-by-base: single nucleotide-level analysis of whole viral genome alignments. BMC Bioinform. 2004;5:96.
10. Bray N, Pachter L. MAVID: constrained ancestral alignment of multiple sequences. Genome Res. 2004;14(4):693–9.
11. Mosig A, Hofacker IL, Stadler PF. Comparative analysis of cyclic sequences: viroids and other small circular RNAs. GCB. 2006;83:93–102.
12. Kawai Y, Saito T, Kitazawa H, Itoh T. Gassericin A; an uncommon cyclic bacteriocin produced by Lactobacillus gasseri LA39 linked at N-and C-terminal ends. Biosci Biotech Biochem. 1998;62(12):2438–40.
13. Kemperman R, Kuipers A, Karsens H, Nauta A, Kuipers O, Kok J. Identification and characterization of two novel clostridial bacteriocins, circularin A and closticin 574. Appl Environ Microbiol. 2003;69(3):1589–97.
14. Weiner J, Bornberg-Bauer E. Evolution of circular permutations in multidomain proteins. Mol Biol Evol. 2006;23(4):734–43.
15. Ponting CP, Russell RB. Swaposins: circular permutations within genes encoding saposin homologues. Trends Biochem Sci. 1995;20(5):179–80.
16. Benson DA, Karsch-Mizrachi I, Lipman DJ, Ostell J, Rapp BA, Wheeler DL. GenBank. Nucleic Acids Res. 2000;28:15–8.
17. Rice P, Longden I, Bleasby A. EMBOSS: the European molecular biology open software suite. Trends Genet. 2000;16(6):276–7.
18. Barton C, Iliopoulos CS, Pissis SP. Fast algorithms for approximate circular string matching. Algorithms Mol Biol. 2014;9:1–10.
19. Barton C, Iliopoulos CS, Pissis SP. Language and automata theory and applications—9th international conference, LATA 2015, Proceedings. In: Dediu AH, Formenti E, Martin-Vide C, Truthe B, editors. Average-case optimal approximate circular string matching, vol. 8977, Lecture notes in computer science. Berlin: Springer; 2015. p. 85–96.
20. Athar T, Barton C, Bland W, Gao J, Iliopoulos CS, Liu C, Pissis SP. Fast circular dictionary-matching algorithm. Math Struct Comput Sci. 2015;FirstView:1–14. doi:10.1017/S0960129515000134.
21. Maes M. On a cyclic string-to-string correction problem. IPL. 1990;35(2):73–8.
22. Marzal A, Barrachina S. Speeding up the computation of the edit distance for cyclic strings. ICPR. 2000;2:891–4.
23. Bunke H, Buhler U. Applications of approximate string matching to 2D shape recognition. Pattern Recognit. 1993;26(12):1797–812.
24. Barton C, Iliopoulos CS, Kundu R, Pissis SP, Retha A, Vayani F. Proceedings of lecture notes in computer science. In: Bampis E, editor. Accurate and efficient methods to improve multiple circular sequence alignment. In experimental algorithms—14th international symposium, SEA, vol. 9125, Berlin: Springer; 2015. p. 247–58.
25. Fernandes F, Pereira L, Freitas AT. CSA: an efficient algorithm to improve circular DNA multiple alignment. BMC Bioinform. 2009;10:1–13.
26. Lee T, Na JC, Park H, Park K, Sim JS. Finding consensus and optimal alignment of circular strings. Theor Comput Sci. 2013;468:92–101.
27. Pisanti N, Giraud M, Peterlongo P. Filters and seeds approaches for fast homology searches in large datasets. In: Elloumi M, Zomaya AY, editors. Algorithms in computational molecular biology. Hoboken: Wiley; 2010. p. 299–320.
28. Ukkonen E. Approximate string-matching with q-grams and maximal matches. Theor Comput Sci. 1992;92:191–211.
29. Burkhardt S, Crauser A, Ferragina P, Lenhof HP, Rivals E, Vingron M. q-gram based database searching using a suffix array (QUASAR). In: RECOMB '99 proceedings of the third annual international conference on Computational molecular biology. New York, NY: ACM; 1999. p. 77–83.
30. Rasmussen K, Stoye J, Myers E. Efficient q-gram filters for finding all epsilon-matches over a given length. J Comput Biol. 2006;13(2):296–308.
31. Peterlongo P, Sacomoto GA, do Lago AP, Pisanti N, Sagot MF. Lossless filter for multiple repeats with bounded edit distance. Algorithm Mol Biol. 2009;4:3. doi:10.1186/1748-7188-4-3.
32. Peterlongo P, Pisanti N, Boyer F, do Lago AP, Sagot MF. Lossless filter for multiple repetitions with hamming distance. JDA. 2008;6(3):497–509.

33. Manber U, Myers EW. Suffix arrays: a new method for on-line string searches. SIAM J Comput. 1993;22(5):935–48.
34. Grossi R, Iliopoulos CS, Mercas R, Pisanti N, Pissis SP, Retha A, Vayani F. Circular sequence comparison with q-grams. In: Pop M, Touzet H, editors. Algorithms in bioinformatics—15th international workshop, WABI 2015, Atlanta, GA, USA, September 10–12, 2015, Proceedings, vol. 9289, Lecture notes in computer science. Berlin: Springer; 2015. p. 203–16.
35. Crochemore M, Hancart C, Lecroq T. Algorithms on strings. New York: Cambridge University Press; 2007.
36. Fischer J. Inducing the LCP-Array. In: Dehne F, Iacono J, Sack J-R, editors. 12th WADS, Volume 6844 of LNCS. 2011. p. 374–85.
37. Ehlers T, Manea F, Mercaş R, Nowotka D. *k*-Abelian pattern matching. In: Shur AM, Volkov MV, editors. 18th DLT, Volume 8633 of LNCS. 2014. p. 178–90.
38. Burcsi P, Cicalese F, Fici G, Lipták Z. Algorithms for jumbled pattern matching in strings. Int J Found Comput Sci. 2012;23(2):357–74.
39. Needleman SB, Wunsch CD. A general method applicable to the search for similarities in the amino acid sequence of two proteins. J Mol Biol. 1970;48(3):443–53.
40. Gotoh O. An improved algorithm for matching biological sequences. J Mol Biol. 1982;162(3):705–8.
41. Smith TF, Waterman MS. Identification of common molecular subsequences. J Mol Biol. 1981;147:195–7.
42. Fletcher W, Yang Z. INDELible: a flexible simulator of biological sequence evolution. Mol Biol Evol. 2009;26(8):1879–88.
43. Gog S, Beller T, Moffat A, Petri M. From theory to practice: plug and play with succinct data structures. In: Gudmundsson J, Katajainen J, editors. 13th international symposium on experimental algorithms, (SEA 2014). 2014. p. 326–37.
44. Wheeler TJ. Large-scale neighbor-joining with NINJA. In: Salzberg S, Warnow TJ, editors. Algorithms in bioinformatics, Springer; 2009. p. 375–89.
45. Robinson D, Foulds LR. Comparison of phylogenetic trees. Math Biosci. 1981;53:131–47.
46. Sukumaran J, Holder MT. DendroPy: a python library for phylogenetic computing. Bioinformatics. 2010;26(12):1569–71.
47. Pruitt KD, Tatusova T, Maglott DR. NCBI reference sequences (RefSeq): a curated non-redundant sequence database of genomes, transcripts and proteins. Nucleic Acids Res. 2007;35(suppl 1):D61–5.
48. Rojas A, Romeu A. A sequence analysis of the β-glucosidase sub-family B. FEBS Lett. 1996;378:93–7.
49. UniProt Consortium. UniProt: a hub for protein information. Nucleic Acids Res. 2015;43(Database issue):D204–12. doi:10.1093/nar/gku989.
50. Wang CK, Kaas Q, Chiche L, Craik DJ. CyBase: a database of cyclic protein sequences and structures, with applications in protein discovery and engineering. Nucleic Acids Res. 2008;36(suppl 1):D206–10.