Pattern Recognition Letters 88 (2017) 81–87

Contents lists available at ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

A faster and more accurate heuristic for cyclic edit distance computation

Lorraine A.K. Ayad[a], Carl Barton[b], Solon P. Pissis[a,*]

[a] Department of Informatics, King's College London, London WC2R 2LS, UK
[b] European Bioinformatics Institute, Wellcome Trust Genome Campus, Hinxton CB10 1SD, UK

ARTICLE INFO

Article history:
Received 20 February 2016
Available online 24 January 2017

MSC:
41A05
41A10
65D05
65D17,

Keywords:
Cyclic edit distance
Circular sequences
Cyclic strings
Chain code
q-gram distance

ABSTRACT

Sequence comparison is the core computation of many applications involving textual representations of data. Edit distance is the most widely used measure to quantify the similarity of two sequences. Edit distance can be defined as the minimal total cost of a sequence of edit operations to transform one sequence into the other; for a sequence x of length m and a sequence y of length n, it can be computed in time $\mathcal{O}(mn)$. In many applications, it is common to consider sequences with circular structure: for instance, the orientation of two images or the leftmost position of two linearised circular DNA sequences may be irrelevant. To this end, an algorithm to compute the cyclic edit distance in time $\mathcal{O}(mn \log m)$ was proposed (Maes, 2003 [18]) and several heuristics have been proposed to speed up this computation. Recently, a new algorithm based on q-grams was proposed for circular sequence comparison (Grossi et al., 2016 [13]). We extend this algorithm for cyclic edit distance computation and show that this new heuristic is faster and more accurate than the state of the art. The aim of this letter is to give visibility to this idea in the pattern recognition community.

1. Introduction

Sequence comparison is a fundamental step in many applications involving textual representations of data. Alignments constitute one of the processes commonly used to compare sequences; they are based on notions of distance or of similarity between strings. Edit distance is the most widely used measure to quantify the similarity (or dissimilarity) of two given sequences. It can be defined as the minimal total cost of a sequence of elementary edit operations to transform one sequence into the other; for a sequence x of length m and a sequence y of length n, it can be computed in time $\mathcal{O}(mn)$ [8].

In many applications it is common to consider sequences with circular structure: for instance, the orientation of two images or the leftmost position of two linearised circular DNA sequences may be *irrelevant*.

In [21], the authors show that computing the edit distance can be used to classify handwritten digits, where the contours of the digits are represented with an 8-direction chain-code [11]; a sequence over an eight-letter alphabet, representing the eight cardinal directions that the contour faces when following the outline of an image in a clockwise motion.

Example applications where image retrieval is required include digital libraries and multimedia editing [28]. Computing the cyclic edit distance is a key requirement for image processing and shape matching. The contours of a shape may be represented through a cyclic sequence which can be used in the computation of the cyclic edit distance. This can identify similarities in shapes which appear to be distinct from one another [20,26].

Circular molecular structures are abundant in all domains of life: bacteria, archaea, and eukaryotes, and in viruses. Exhaustive reviews of circular molecular structures can be found in [7] and [14].

Using standard techniques to align circular sequences could incorrectly yield a high genetic distance between closely-related sequences. Indeed, when sequencing molecules, the position where a circular sequence starts can be totally arbitrary. For instance, the linearised human (NC 001807) and chimpanzee (NC 001643) mitochondrial DNA (mtDNA) sequences do not start in the same region [13]. Due to this arbitrariness, a suitable rotation of one sequence would give much better results for a pairwise alignment. This motivates the design of efficient algorithms that are specifically devoted to the comparison of circular sequences [1,4,5,13].

* Corresponding author.
E-mail address: solon.pissis@kcl.ac.uk (S.P. Pissis).

http://dx.doi.org/10.1016/j.patrec.2017.01.018

The *cyclic edit distance* (CED) problem can be defined as follows. Given a sequence x of length m and a sequence y of length n, find the minimal edit distance between any conjugate (cyclic rotation) of x and any conjugate of y.

Few exact algorithms exist which are able to compute the cyclic edit distance between x and y. Maes designed an elegant divide-and-conquer algorithm which runs in time $\mathcal{O}(mn \log m)$ [18]. The idea of this algorithm is to identify optimal edit paths which do not cross each other on the edit graph of xx and y. An exact branch and bound algorithm based on Maes's algorithm, which runs in time $\mathcal{O}(mn \log m)$, was proposed by Barrachina and Marzal [2]. This method explores only the nodes on the edit graph that could lead to an optimal path, resulting in a much faster algorithm *on average*.

Several heuristic approaches exist for approximating the cyclic edit distance. One of the first ones is the Bunke and Buhler (BBA) algorithm [6]. It estimates a lower bound for the cyclic edit distance by searching for an optimal path in time $\mathcal{O}(mn)$. The extended Bunke and Buhler method (EBBA) computes an estimation of the upper bound for the exact cyclic edit distance, also in time $\mathcal{O}(mn)$ [22]. The weighted Bunke and Buhler algorithm (WeBBA) combines the lower and upper bound estimations, computed by the BBA and EBBA algorithms, to produce an approximation of the cyclic edit distance in time $\mathcal{O}(mn)$ [23]. It is perhaps the best performing heuristic currently.

Palazon-Gonzalez and Marzal [27] studied the same problem but from the *indexing* point of view for classification and retrieval. Their methods eliminate searching for a distance when it is known that it will be greater than the distance (external bound) to the nearest neighbour. They propose two algorithms. The first one modifies the branch and bound algorithm of Barrachina and Marzal [2] by avoiding exploring ranges known to be lower than the lower bound in the branch and bound computation. The second one modifies the BBA algorithm by preventing searching for distances when it is known that the final result will not improve the current external bound.

Our contribution. In this letter, we propose hCED, a new heuristic algorithm for cyclic edit distance computation. The first important step of this computation is based on an idea that has not been explored by the previous heuristics; that is, considering q-grams, factors of length q. Informally, the q-gram similarity, defined as a distance in [29], is the number of q-grams shared by the two sequences. Theoretical insight to support the suitability of the algorithm is provided. hCED can be split into the following three main stages:

1. The rotation of x that minimises a generalisation of the q-gram distance between x and y is computed using the algorithm in [13];
2. A refinement on this rotation of x is carried out by examining only some short prefixes and suffixes of the rotation and sequence y;
3. Finally, the edit distance between the refined rotation of x and sequence y is computed.

Our main contribution is an extensive experimental study using both DNA and 8-direction chain-code datasets. These results show that hCED is generally faster, up to one order of magnitude, and more accurate than existing state-of-the-art heuristics. A free open-source implementation of hCED is also made available as opposed to current methods.

2. Definitions

We begin with a few definitions, following Crochemore et al. [8]. 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 $y = uxv$. 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.

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 $x^0 = x$. Consider, for instance, the string $x = x^0 = \texttt{ababbbc}$; which has the following rotations: $x^1 = \texttt{bababbca}$, $x^2 = \texttt{ababbcab}$, and so on. We say that two strings x and y are *conjugate* if there exist two strings u and v such that $x = uv$ and $y = vu$.

Given a string x of length m and a string y of length $n \ge m$, the *edit distance*, denoted by $\delta_E(x, y)$, is defined as the minimal total cost of edit operations required to transform one string into the other. In general, the allowed operations are as follows:

- *Insertion*: insert a letter in y, not present in x; $(\varepsilon, b), b \neq \varepsilon$
- *Deletion*: delete a letter in y, present in x; $(a, \varepsilon), a \neq \varepsilon$
- *Substitution*: replace a letter in y with a letter in x; (a, b), $a \neq b$, and $a, b \neq \varepsilon$.

By $\mathsf{ins}(b)$, $\mathsf{del}(a)$, and $\mathsf{sub}(a, b)$, $a \neq b$, and $a, b \in \Sigma$, we denote the cost of insertion, deletion, and substitution operations, respectively. In many applications, we only want to count the number of edit operations, considering the cost of each to be 1 [17]. This distance is known as *Levenshtein distance*, a special case of edit distance where unit costs apply.

The *cyclic edit distance*, denoted by $\delta_{\mathrm{CE}}(x, y)$, is defined as $\delta_{\mathrm{CE}}(x, y) = \min_i(\min_j \delta_E(x^i, y^j)) = \min_i \delta_E(x^i, y)$ [18].

We give some further definitions following Ukkonen [29]. 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.

Definition 1. Given two strings x and y and an integer $q > 0$, the *q-gram distance* $D_q(x, y)$ is defined as

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

Jokinen and Ukkonen [15] showed the following bound which is directly applicable to the Levenshtein distance.

Lemma 1 [15]**.** *Let x and y be strings with Levenshtein distance k. Then at least $|x| + 1 - (k+1)q$ of the $|x| - q + 1$ q-grams of x occur in y.*

For a given integer parameter $\beta \ge 1$, Grossi et al. [13] defined a generalisation of the q-gram distance 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 this letter, 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 2. 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

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

3. Algorithm hCED

We first begin by extending Lemma 1 to non-unit costs.

Lemma 2. *Let x and y be strings with edit distance k, such that $C = \min\{\mathsf{ins}(b), \mathsf{del}(a), \mathsf{sub}(a,b)\}$, for some $C > 1$, $a \neq b$, and $a, b \in \Sigma$. Then at least $|x| + 1 - (\lfloor k/C \rfloor + 1)q$ of the $|x| - q + 1$ q-grams of x occur in y.*

Proof. By assumption we have that $\delta_E(x, y) = k$, and if the edit operations do not have uniform cost, we have that the number of edit operations is less than or equal to $\lfloor k/C \rfloor$. Each edit operation could alter at most q different q-grams and hence the lemma follows. □

Consider the case when $C = \min\{\mathsf{ins}(b), \mathsf{del}(a), \mathsf{sub}(a,b)\}$, $D = \max\{\mathsf{ins}(b), \mathsf{del}(a), \mathsf{sub}(a,b)\}$, and $D - C = \mathcal{O}(1)$. This assumption captures most, if not all, real-world edit-distance-based applications. We claim that the lower bound on the number of q-grams is *good* in the following sense. The number e of edit operations must be $\lfloor k/D \rfloor \leq e \leq \lfloor k/C \rfloor$. For $|x| = |y|$ and $e = (|x| - q + 1)/q$, it is easy to design a string such that each operation alters exactly q q-grams. We can then see that the best bound we can achieve in the above lemma, without some stronger assumptions, is $|x| + 1 - (\lfloor k/D \rfloor + 1)q$ shared q-grams and therefore in such cases, the bound in Lemma 2 is within a constant factor. Note that the choice of $e = (n - q + 1)/q$ is not arbitrary; should e be more than this, the pigeon-hole principle shows that it is not possible to distribute e operations in such a way that each occurs at least q positions apart. This means that each operation can no longer alter exactly q q-grams.

Lemma 3. *Let x and y be two conjugate strings. For a given q, x and y share at most $|x| - q + 1$ q-grams and at least $|x| - 2q + 2$.*

Proof. The first part is trivial. Consider the case when x is a string with a distinct letter per position and $q = 1$. Then x and y share exactly $|x| - q + 1$ distinct q-grams.

For the second part, and by definition, notice that x and y can always be decomposed to x_1x_2 and y_1y_2, respectively, where x_1, x_2, y_1, y_2 are strings, such that $x_1 = y_2$ and $x_2 = y_1$. Then it is not difficult to see that by choosing an appropriate decomposition, each pair, (x_1, y_2) and (x_2, y_1), shares $|x_1| - q + 1$ and $|x_2| - q + 1$ q-grams, respectively. The sum $|x_1| - q + 1 + |x_2| - q + 1$ can be rewritten as $|x_1| - q + 1 + (|x| - |x_1|) - q + 1$ which gives $|x| - 2q + 2$. This concludes the proof. □

The small difference of the two bounds shown in Lemma 3 tells us that the q-gram distance is not a *good* distance by itself to recover the rotation of x that minimises the edit distance to y.

Based on the aforementioned remarks, we proceed with designing hCED, a three-stage heuristic algorithm for cyclic edit distance computation. In the first stage, an initial rotation of x is computed using the β-blockwise q-gram distance. In the second stage, a refinement of this rotation is performed; and finally, the edit distance between this refined rotation of x and string y is computed.

3.1. Stage 1: circular sequence comparison with q-grams

Grossi et al. [13] presented an exact algorithm to compute the β-blockwise q-gram distance between x and y. The algorithm is based on constructing the suffix array [19] for string xxy and assigning a rank to the prefix with length q of each suffix with length at least q, based on its order in the suffix array. The algorithm then finds the rotation i of x such that the β-blockwise q-gram distance between x^i and y is minimal. Ties are broken arbitrarily. The algorithm runs in time and space $\mathcal{O}(\beta m + n)$. The first stage of hCED is essentially the aforementioned algorithm that exploits q-grams (see Lemma 2). For $\beta = 1$ this corresponds to minimising the standard q-gram distance which is not satisfactory (see Lemma 3); however, the generalisation to β blocks enforces the property of locality.

3.2. Stage 2: refinement

In the second stage, hCED refines rotation x^i and produces a refined rotation, denoted by x^r. When in the first stage, the algorithm splits strings x and y into β blocks, it naturally disregards locality within each block. Thus when the initial rotation is produced, it may need to be shifted again slightly to the left or to the right. To this end, we introduce a new input parameter $0 < P \leq \beta/3$ which defines the length $L = \lfloor P \times \frac{m}{\beta} \rfloor$ of the prefixes and suffixes of x^i and y to be considered by the refinement.

The algorithm proceeds as follows. It creates two new strings x' and y' both of length $3L$. In particular, x' is of the form $x_0^i x_1^i x_2^i$, where x_0^i is the prefix of length L of string x^i; x_1^i is a string of length L consisting only of a letter \$ $\notin \Sigma$; and x_2^i is the suffix of length L of string x^i. The same is done for y using the prefix and suffix of y, resulting in the new string y'.

Each rotation of x' is then compared to y' excluding when a letter of x_1^i (letter \$) is found at index 0 of the rotation of x'. Notice that the notion of edit distance is not appropriate here due to the existence of letter \$ which denotes a *don't care* letter. We thus rather utilise a notion of similarity between strings, for which equalities between letters are positively valued; inequalities, insertions, and deletions are negatively valued; and comparisons involving letter \$ are neither positively nor negatively valued. The search consists then in *maximising* a quantity representative of the similarity between the strings.

To this end we make use of the Needleman–Wunsch algorithm [25] to compute a similarity score for each rotation of string x' and string y'. The rotation of x' which results in the maximum score is chosen as the best rotation, and hence, the final rotation x^r of x is computed based on this rotation of x'. Ties are broken arbitrarily. Both x' and y' have length $3L$ resulting in a single Needleman–Wunsch call to have a running time of $\mathcal{O}(L^2)$. As this computation is done exactly $2L$ times, once for each rotation, the overall computation of the refinement takes time $\mathcal{O}(L^3)$.

3.3. Stage 3: edit distance computation

In the third stage, hCED computes the edit distance between strings x^r and y. Myers bit-vector algorithm is used to compute the edit distance when using unit costs for insertion, deletion, and substitution [24]. Myers algorithm runs in time $\mathcal{O}(\lceil \frac{m}{w} \rceil n)$, where w is the word size of the machine. The standard edit distance algorithm is used when computing the edit distance with non-unit costs. It runs in time $\mathcal{O}(mn)$ [8]. Hence, notice that, compared to the other heuristics, hCED offers an additional advantage. If one is only interested in the rotation minimising the cyclic edit distance, but not the actual value of the distance, they can use algorithm hCED and skip the third stage, allowing for a much faster computation.

3.4. Analysis

The first two stages of algorithm hCED run in total time $\mathcal{O}(\beta m + n + L^3)$. The parameters β and P can be tailored depending on the application; however our experiments show that setting $\beta = \mathcal{O}(m^{1/3})$ and $P = \mathcal{O}(1)$ performs very well in applications with circular strings. This can be theoretically explained as follows. Notice that β should not be too large to allow some flexibility corresponding to insertions and deletions in the alignment. For P, this is not surprising either as the rationale of the second stage is to refine via examining *only* the leftmost and rightmost blocks of each

sequence. These parameter choices imply that the first two stages run in time $\mathcal{O}(m^2+n)$. The third stage runs in time $\mathcal{O}(\lceil \frac{m}{w} \rceil n)$ with unit costs and in time $\mathcal{O}(mn)$ with non-unit costs. The space complexity is $\mathcal{O}(\beta m+n)$; the edit distance and Needleman–Wunsch algorithms can both be implemented in $\mathcal{O}(m+n)$ space [8].

4. Experimental results

Algorithm hCED was implemented in C++ as a program to compute an approximation of the cyclic edit distance. Given two sequences x and y in MultiFASTA format, the number of β blocks, and the length q of the q-grams, hCED finds an approximation of the rotation of x that minimises its edit distance from y. It can also output the corresponding rotation of x. The implementation is distributed under the GNU General Public License (GPL), and it is available freely at http://github.com/lorrainea/hCED.

Another program[1] was used to produce experimental results for the Maes, Branch and Bound, BBA, EBBA, and WeBBA algorithms and compare their performance against that of algorithm hCED. The experiments were conducted on a computer using an Intel Core i3-5005U CPU at 2.00 GHz under GNU/Linux. Both programs were compiled with g++ version 4.8.5 at optimisation level 3 (O3). All input datasets referred to in this section are publicly maintained at the same website.

Myers bit-vector algorithm was implemented using the SeqAn library [9]. The standard edit distance algorithm was also implemented to show how the hCED algorithm compares to the other heuristics when both unit and non-unit costs are used for the edit distance operations.

4.1. Synthetic data

DNA datasets were simulated using INDELible [10], which produces sequences in a (Multi)FASTA file. A rate for insertions, deletions, and substitutions are defined by the user to vary the similarity of the sequences. 8-direction chain-code datasets were also generated using a simple script that generates random (uniform distribution) sequences over $\Sigma = \{0, 1, \ldots, 7\}$. All datasets used in the experiments are denoted in the form *A.B.C*, where *A* represents the number of sequences in the dataset; *B* the average length of the sequences; and *C* the percentage of dissimilarity between the sequences. The dissimilarity values of 5, 20, and 35 were used for both the DNA data and chain-codes.

Nine datasets were simulated to measure the accuracy for DNA sequences. Each dataset had a varying number of sequences, all with an average length of 2500. For each dataset, the algorithms were run for every possible pair of sequences in the set. Three 8-direction chain-code datasets were also produced. These datasets consisted of twelve sequences in each set with an average length of 500. Similarly, for each dataset, the algorithms were run for every possible pair of sequences in the set. For all datasets, we made use of the following parameter values for algorithm hCED: $q = 5$, $\beta = m/50$, and $P = 1$.

The results in Table 1 show the accuracy of the algorithms for both data types when unit costs were used for insertion, deletion and substitution. In some applications, in particular in bioinformatics, the cost for insertions and deletions is set higher than the cost for substitutions. Table 2 shows the accuracy for each of the data types when non-unit costs of 3, 3, and 1 were used for insertion, deletion, and substitution, respectively. It becomes evident from these results that algorithm hCED is the most accurate in comparison to the other heuristic algorithms.

To measure the time performance for both data types, seven pairs of sequences of varying length were simulated. The running time for all sequence pairs were computed ten times and the average was taken. For these experiments, the following parameter values for algorithm hCED were used: $q = 5$, $\beta = \min(50, \sqrt{m})$, and $1 \leq P \leq 2$. Fig. 1 shows the time performance of hCED when using unit costs compared to the other heuristic and exact algorithms. It is clear that as the sequence length grows, hCED is an order of magnitude faster than WeBBA, the current fastest performing algorithm. Notice that hCED is three orders of magnitude faster than Maes's algorithm.

Fig. 2 shows the time performance of algorithm hCED for both data types when using non-unit costs. A comparison between hCED and the other cyclic edit distance algorithms can be seen in the figure. As the sequence length grows, hCED and WeBBA become the fastest performing algorithms. Both figures also show the time performance of algorithm hCED when *only* the rotation is computed: the cyclic edit distance value is *not* computed. It is evident that dismissing the computation of the cyclic edit distance greatly improves the time performance of hCED. The results of hCED confirm our theoretical analysis.

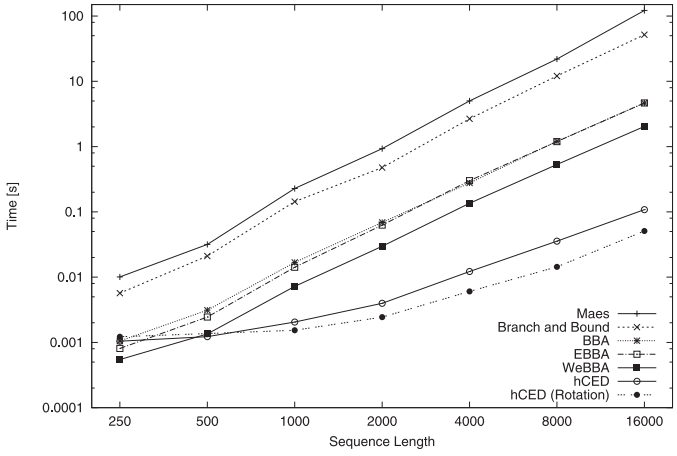
Table 1
Accuracy of heuristic algorithms in comparison to exact results produced by Maes's algorithm for datasets using unit costs. The highest accuracy for each dataset is shown in bold.

DNA				
Accuracy (%)				
Dataset	hCED	BBA	WeBBA	EBBA
12.2500.5	**100.000**	83.302	**100.000**	**100.000**
12.2500.20	**100.000**	76.043	99.939	99.905
12.2500.35	**100.000**	77.673	99.933	99.889
25.2500.5	**100.000**	84.798	99.997	99.968
25.2500.20	**99.975**	74.606	99.903	99.868
25.2500.35	**99.961**	73.478	99.882	99.849
50.2500.5	**100.000**	85.303	99.999	99.960
50.2500.20	**99.999**	79.903	99.977	99.940
50.2500.35	**99.981**	74.043	99.910	99.867
8-direction chain-code				
Accuracy (%)				
Dataset	hCED	BBA	WeBBA	EBBA
12.500.5	**100.000**	82.511	99.895	99.401
12.500.20	**100.000**	81.344	99.718	99.481
12.500.35	**100.000**	87.364	99.783	99.586

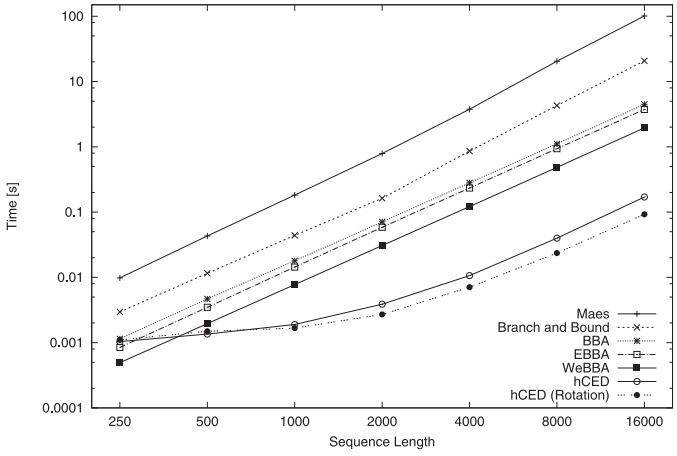
Table 2
Accuracy of heuristic algorithms in comparison to exact results produced by Maes's algorithm for datasets using non-unit costs. The highest accuracy for each dataset is shown in bold.

DNA				
Accuracy (%)				
Dataset	hCED	BBA	WeBBA	EBBA
12.2500.5	**100.000**	79.476	**100.000**	**100.000**
12.2500.20	**99.958**	61.197	99.793	99.763
12.2500.35	**99.997**	73.360	99.953	99.909
25.2500.5	**99.986**	80.950	99.981	99.956
25.2500.20	**99.970**	70.523	99.903	99.864
25.2500.35	**99.942**	65.091	99.865	99.831
50.2500.5	**99.996**	81.131	99.992	99.955
50.2500.20	**99.987**	75.358	99.972	99.937
50.2500.35	**99.969**	69.339	99.932	99.888
8-direction chain-code				
Accuracy (%)				
Dataset	hCED	BBA	WeBBA	EBBA
12.500.5	**99.967**	38.044	99.677	99.282
12.500.20	**99.175**	44.569	98.859	98.592
12.500.35	**99.771**	57.554	99.082	98.854

[1] Obtained through personal communication with author – Guillermo Peris.

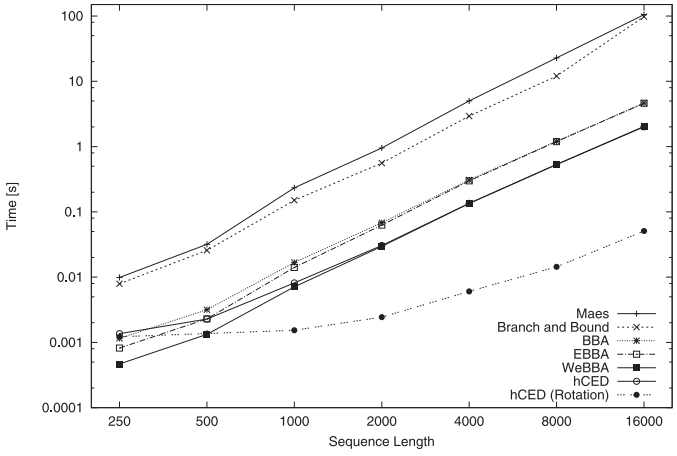


(a) DNA

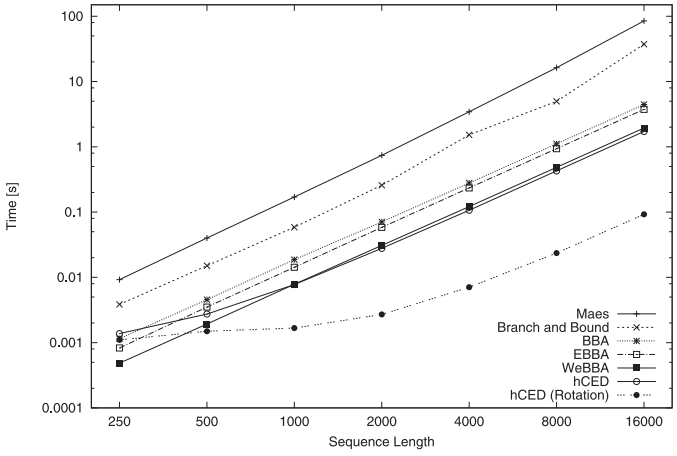


(b) 8-direction chain-code

Fig. 1. Elapsed time to execute datasets using unit costs.



(a) DNA



(b) 8-direction chain-code

Fig. 2. Elapsed time to execute datasets using non-unit costs.

4.2. Real data

Three datasets made up of nucleotide sequences were used to test the hCED algorithm's ability to identify accurate rotations. The first dataset (Mammals) includes 12 mtDNA sequences of mammals, the second dataset (Primates) includes 16 mtDNA sequences of primates, and the last one (Viroids) includes 18 viroid RNA sequences. The average sequence length for Mammals is 16,777 base pairs (bp), for Primates is 16,581 bp, and for Viroids is 363 bp.

Table 3 shows the accuracy of hCED's computation of the cyclic edit distance for each pair, which we denote by AP, in comparison to the other heuristics, as well as the average time taken to do so. The experiment was carried out when using unit costs for insertions, deletions, and substitutions, as well as when using the same non-unit costs previously presented. For the Mammals and Primates datasets, we made use of the following parameter values for algorithm hCED: $q = 5$, $\beta = m/100$, and $P = 1$. For the Viroids dataset, in which the sequences are much shorter, the following parameters were used instead: $q = 5$, $\beta = m/25$, and $P = 1$. It is evident from Table 3, that not only does hCED give the most accurate results, but it is also faster for sequences of long length when using both unit and non-unit costs.

Handwritten digits from the MNIST database [16] were also used and sorted into ten sets. Each image was placed in one of ten datasets, depending on the value of the drawn digit. Each handwritten digit was in the form of a 28 × 28 matrix consisting of pixel values. 5000 of the 60,000 images were extracted and converted into binary matrices. A normalised 8-direction chain-code was produced for the handwritten digits, where a subset can be found in Fig. 3. Normalising the chain-code allows the image to be treated as a circular sequence of minimum magnitude. This produces a sequence independent of the rotation of the image. This was calculated by identifying the number of direction changes be-

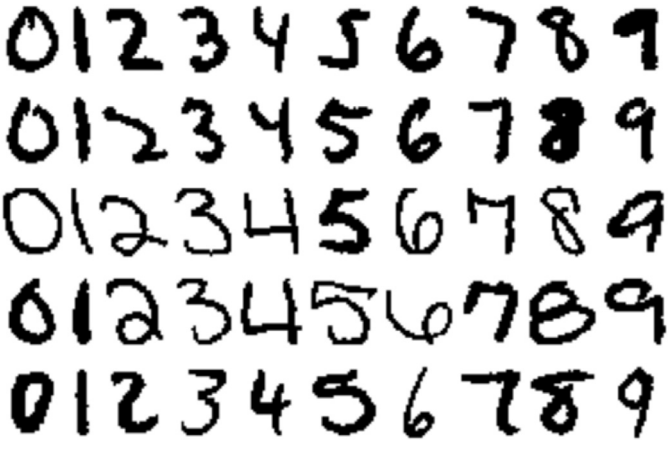


Fig. 3. Handwritten digits.

Table 3
Accuracy of heuristic algorithms in comparison to exact results produced by Maes's algorithm and elapsed-time comparison for real nucleotide data. The highest accuracy and fastest time for each dataset are shown in bold.

Unit costs								
Program	hCED		BBA		WeBBA		EBBA	
Dataset	AP (%)	Time (s)	AP (%)	Time (s)	AP (%)	Time (s)	AP (%)	Time (s)
Mammals	**99.618**	**0.479**	69.477	4.870	96.482	2.162	96.469	5.015
Primates	**99.743**	**0.202**	74.256	4.766	98.749	2.115	98.742	4.904
Viroids	**98.363**	0.003	61.057	0.002	97.874	**0.001**	97.614	0.002
Non-unit costs								
Program	hCED		BBA		WeBBA		EBBA	
Dataset	AP (%)	Time (s)	AP (%)	Time (s)	AP (%)	Time (s)	AP (%)	Time (s)
Mammals	**98.221**	**2.092**	57.092	4.870	86.728	2.202	86.716	5.012
Primates	**99.672**	**1.964**	65.859	4.748	94.443	2.175	94.431	4.898
Viroids	**98.155**	0.003	49.746	0.002	97.623	**0.001**	97.288	0.002

Table 4
Accuracy of heuristic algorithms in comparison to exact results produced by Maes's algorithm for handwritten digits. The highest accuracy for each dataset is shown in bold.

Accuracy (%) - Unit costs				
Dataset	hCED	BBA	WeBBA	EBBA
0.479.55	**91.781**	52.908	90.126	88.271
1.563.43	**93.159**	58.589	92.037	89.629
2.488.73	**94.504**	54.442	90.265	88.860
3.493.80	**95.371**	54.043	89.441	88.277
4.535.69	**93.855**	59.600	90.351	89.022
5.434.78	**95.287**	53.118	87.869	86.640
6.501.60	**94.139**	54.796	88.954	87.328
7.550.65	**92.436**	55.092	88.984	88.485
8.462.56	**94.006**	55.841	90.720	89.023
9.495.54	**94.211**	55.946	89.814	88.029
Accuracy (%) - Non-unit costs				
Dataset	hCED	BBA	WeBBA	EBBA
0.479.55	**87.699**	41.323	80.256	78.535
1.563.43	**88.213**	49.656	85.895	83.727
2.488.73	**89.395**	41.645	77.193	76.145
3.493.80	**87.396**	38.844	72.854	71.849
4.535.69	**89.163**	45.332	77.621	76.350
5.434.78	**85.338**	37.622	69.817	68.870
6.501.60	**86.279**	38.630	72.112	70.759
7.550.65	**88.276**	41.018	75.323	74.108
8.462.56	**87.597**	41.948	75.948	74.398
9.495.54	**86.950**	41.765	76.159	74.715

tween two adjacent elements of the chain-code in an anticlockwise direction (see [12], for details).

Table 4 shows the results of using algorithm hCED to compute the cyclic edit distance for successive pairs in each dataset. Each dataset is in the form *D.E.F*, where *D* represents the drawn digit; *E* the number of sequences in the set; and *F* the average length of the sequences in the set. For these datasets, we made use of the following parameter values for algorithm hCED: $q = 5$, $7 \leq m/\beta \leq 15$, depending on the average length of the sequence, and $P = 1$. It is evident from Table 4, that for all sets, hCED is the most accurate when using both unit and non-unit costs. Running times are not presented for the handwritten digits datasets as the sequence lengths are very small.

5. Conclusion

In this letter, algorithm hCED, a new heuristic approach to approximate the cyclic edit distance, was presented. It is an extension of the q-gram based algorithm presented in [13] adapted for cyclic edit distance computation. Our main contribution is an extensive experimental study to compare hCED against existing state-of-the-art heuristics for the same problem. In particular, we showed that the performance of hCED, in terms of accuracy and speed, outperforms existing heuristics using both DNA and 8-direction chain-code data.

The inherent structure of hCED allows for two important properties: (i) hCED enables the user to compute only the rotation of x (or an approximation of it) that minimises the cyclic edit distance from y, performing even faster if the actual value for the cyclic edit distance is not required; and (ii) this also enables the usage of the fastest known algorithm for the edit distance computation with unit costs if the actual value for the cyclic edit distance is required. Fig. 1 greatly reflects these advantages in practical terms.

Our improvements are particularly important for image retrieval and molecular biology applications. For instance, algorithm hCED can now be directly used for computing the cyclic edit distance between all pairs of sequences for progressive multiple circular sequence alignment [3].

Acknowledgements

We would like to acknowledge The Engineering and Physical Sciences Research Council for their financial support [EP/M50788X/1] for L.A.K.A, as well as Guillermo Peris (Universitat Jaume I) who kindly provided us with the implementation of the competing algorithms.

References

[1] T. Athar, C. Barton, W. Bland, J. Gao, C.S. Iliopoulos, C. Liu, S.P. Pissis, Fast circular dictionary-matching algorithm, Math. Struct. Comput. Sci. (2015) 1–14. First View.
[2] S. Barrachina, A. Marzal, Speeding up the computation of the edit distance for cyclic strings, in: Proceedings. 15th International Conference on Pattern Recognition, 2000, 2, 2000, pp. 891–894.
[3] C. Barton, C.S. Iliopoulos, R. Kundu, S.P. Pissis, A. Retha, F. Vayani, Accurate and efficient methods to improve multiple circular sequence alignment, in: E. Bampis (Ed.), Experimental Algorithms - 14th International Symposium, SEA 2015, Proceedings, vol. 9125 of Lecture Notes in Computer Science, Springer, 2015, pp. 247–258.
[4] C. Barton, C.S. Iliopoulos, S.P. Pissis, Fast algorithms for approximate circular string matching, Algorithms Mol. Biol. 9 (2014) 9.
[5] C. Barton, C.S. Iliopoulos, S.P. Pissis, Average-case optimal approximate circular string matching, in: A.H. Dediu, E. Formenti, C. Martin-Vide, B. Truthe (Eds.), Language and Automata Theory and Applications - 9th International Conference, LATA 2015, Proceedings, vol. 8977 of Lecture Notes in Computer Science, Springer, 2015, pp. 85–96.
[6] H. Bunke, U. Buhler, Applications of approximate string matching to 2d shape recognition, Pattern Recognit. 26 (1993) 1797–1812.
[7] D.J. Craik, N.M. Allewell, Thematic minireview series on circular proteins, J. Biol. Chem. 287 (32) (2012) 26999–27000.

[8] M. Crochemore, C. Hancart, T. Lecroq, Algorithms on Strings, Cambridge University Press, New York, NY, USA, 2007.
[9] A. Doring, D. Weese1, T. Rausch1, K. Reinert, SeqAn an efficient, generic C++ library for sequence analysis, BMC Bioinformatics (2008) 1–9.
[10] W. Fletcher, Z. Yang, INDELible: a flexible simulator of biological sequence evolution, Mol. Biol. Evol. 8 (2009) 1879–1888.
[11] H. Freeman, On the encoding of arbitrary geometric configurations, Electron. Comput. IRE Trans. EC-10 (1961) 260–268.
[12] R.C. Gonzalez, R.E. Woods, Digital Image Processing (2nd Edition), Prentice Hall, 2002.
[13] R. Grossi, C.S. Iliopoulos, R. Mercas, N. Pisanti, S.P. Pissis, A. Retha, F. Vayani, Circular sequence comparison: algorithms and applications, Algorithms Mol. Biol. 11 (2016) 12.
[14] D.R. Helinski, D.B. Clewell, Circular DNA, Annu. Rev. Biochem. 40 (1971) 899–942.
[15] P. Jokinen, E. Ukkonen, Two algorithms for approximate string matching in static texts, in: MFCS, 1991, pp. 240–248.
[16] Y. LeCun, C. Cortes, The MNIST database of handwritten digits(1999). http://yann.lecun.com/exdb/mnist/.
[17] V.I. Levenshtein, Binary Codes Capable of Correcting Deletions, Insertions, and Reversals, Technical Report, 1966.
[18] M. Maes, On a cyclic string-to-string correction problem, Inf. Process Lett. 35 (2003) 73–78.
[19] U. Manber, E.W. Myers, Suffix arrays: a new method for on-line string searches, SIAM J. Comput. 22 (5) (1993) 935–948.
[20] A. Marzal, V. Palazon-Gonzalez, Dynamic time warping of cyclic strings for shape matching, in: Pattern Recognition and Image Analysis, vol. 3687 of Lecture Notes in Computer Science, Springer, 2005, pp. 644–652.
[21] A. Marzal, E. Vidal, Computation of normalized edit distance and applications, Pattern Anal. Machine Intell. IEEE Trans. 15 (1993) 926–932.
[22] R.A. Mollineda, E. Vidal, F. Casacuberta, Efficient techniques for a very accurate measurement of dissimilarities between cyclic patterns, Adv. Pattern Recogn. 1876 (2000) 337–346.
[23] R.A. Mollineda, E. Vidal, F. Casacuberta, Cyclic sequence alignments: approximate versus optimal techniques, Int. J. Pattern Recognit Artif Intell. 16 (2002) 291–299.
[24] G. Myers, A fast bit-vector algorithm for approximate string matching based on dynamic programming, J. ACM 46 (1999) 395–415.
[25] S.B. Needleman, C.D. Wunsch, A general method applicable to the search for similarities in the amino acid sequences of two proteins, J. Mol. Biol. 48 (1970) 443–453.
[26] V. Palazon-Gonzalez, A. Marzal, On the dynamic time warping of cyclic sequences for shape retrieval, Image Vis. Comput. 30 (2012) 978–990.
[27] V. Palazon-Gonzalez, A. Marzal, Speeding up the cyclic edit distance using laesa with early abandon, Pattern Recogn. Lett. 62 (2015) 1–7.
[28] T. Sikora, The MPEG-7 visual standard for content description-an overview, Circuits Syst. Video Technol. IEEE Trans. 11 (2001) 696–702.
[29] E. Ukkonen, Approximate string-matching with q-grams and maximal matches, Theor. Comput. Sci. 92 (1992) 191–211.