Fast Optimal Circular Clustering and Applications on Round Genomes

Tathagata Debnath and Mingzhou Song*

Abstract—Round genomes are found in bacteria, plant chloroplasts, and mitochondria. Genetic or epigenetic marks can present biologically interesting clusters along a circular genome. The circular data clustering problem groups N points on a circle into K clusters to minimize the within-cluster sum of squared distances. Repeatedly applying the K-means algorithm takes quadratic time, impractical for large circular datasets. To overcome this issue, we developed a reproducible fast optimal circular clustering (FOCC) algorithm of worst-case $\mathcal{O}(KN\log^2 N)$ time. The core is a fast optimal framed clustering algorithm, which we designed by integrating two divide-and-conquer and one bracket dynamic programming strategies. The algorithm is optimal based on a property of monotonic increasing cluster borders over frames on linearized data. On clustering 50,000 circular data points, FOCC outruns brute-force or heuristic circular clustering by three orders of magnitude in time. We produced clusters of CpG sites and genes along three round genomes, exhibiting higher quality than heuristic clustering. More broadly, the presented subquadratic-time algorithms offer the fastest known solution to not only framed and circular clustering, but also angular, periodical, and looped clustering. We implemented these algorithms in the R package 'OptCirClust' (https://CRAN.R-project.org/package=OptCirClust).

Index Terms—Circular Clustering, Framed Clustering, CpG Island, Round Genome, Mitochondria, Bacteria.

1 Introduction

Round genomes widely exist in living systems [1] such as bacteria [2], chloroplasts in plants [3], and mitochondria in eukaryotes [4]. They are no less abundant than linear genomes, as the number of bacterial species is in the order of millions [5]. Circular RNA [6] and extrachromosomal circular DNA [7] molecules are linked to multiple diseases including cancer. Genomes are uneven [8], [9], whose elements are not uniformly spread throughout a chromosome. Clusters of CpG islands [10], gene locations [11], and origin of replication [12] can point to active regions [13] or hot spots [14], [15] along a circular molecule.

The circular clustering problem takes N points on a circle as input and generates K clusters as output. Unlike linear univariate clustering [16], [17], there is no starting or ending position in circular data. An intuitive solution to circular clustering is to consider all possible starting positions of circular data to form frames and repeatedly apply a K-means algorithm [18] on each frame. The asymptotic runtime of such an approach is $\mathcal{O}(N^2 t)$, where t is the number of iterations in each run of the K-means algorithm. Not only slow, this approach is not necessarily reproducible due to stochastic behavior of heuristic K-means methods.

Although the multivariate clustering problem is *NP*-hard, univariate clustering is exactly solvable with reproducibility by dynamic programming in polynomial time [19], [20], [21]. Recent work by Song and Zhong [21] provides a low-overhead method to achieve optimal univariate clustering in $\mathcal{O}(KN)$ time on sorted linear data. However, repeatedly applying optimal univariate clustering starting at each circular point will take a quadratic runtime of $\mathcal{O}(KN^2)$, which is still impractical for a circular genome with millions of base pairs.

To overcome the inefficiency of a simple extension of K-means, we present a fast optimal circular clustering (FOCC) algorithm, which runs in the worst-case $\mathcal{O}(KN\log^2 N)$ time. The FOCC algorithm integrates two divide-and-conquer and one bracket dynamic programming strategies to drastically cut down runtime. It first arranges the circular data O with N points into linear data X with $2N-1$ points, by traversing through the sorted circular data twice. Figure 1(a) and (b) illustrates this conversion. Then we introduce a fast optimal framed clustering algorithm to examine each data frame of size N along the linearized data to identify an optimal frame. Each frame is marked by an ID, ranging from 0 to $N-1$, which is the smallest index to points in the frame. The optimality of framed clustering is guaranteed based on a property of monotonic increasing cluster borders over frames on linearized data. Figure 1(b) illustrates framed clustering on linearized data to constrain the search of optimal cluster borders by those of two nearby frames. Figure 1(c) shows the effect of search space reduction on the dynamic programming matrices of framed clustering. The output clustering of FOCC is illustrated in Figure 1(d). On a circular genome of 50,000 events, the FOCC algorithm empirically runs three-orders-of-magnitude faster than brute-force or heuristic circular clustering. The advantage in optimality becomes evident as the number of clusters increases.

The main contributions of this work are as follows:

- We establish the fast circular clustering algorithm FOCC based on optimal framed clustering that mas-

- T. Debnath is with the Department of Computer Science, New Mexico State University, Las Cruces, NM, 88003, USA.
E-mail: tirtha.debnath@gmail.com, tathadbn@nmsu.edu
- M. Song is with the Department of Computer Science and Graduate Program in Molecular Biology and Interdisciplinary Life Sciences, New Mexico State University, Las Cruces, NM, 88003, USA.
E-mail: joemsong@cs.nmsu.edu

*: Corresponding author

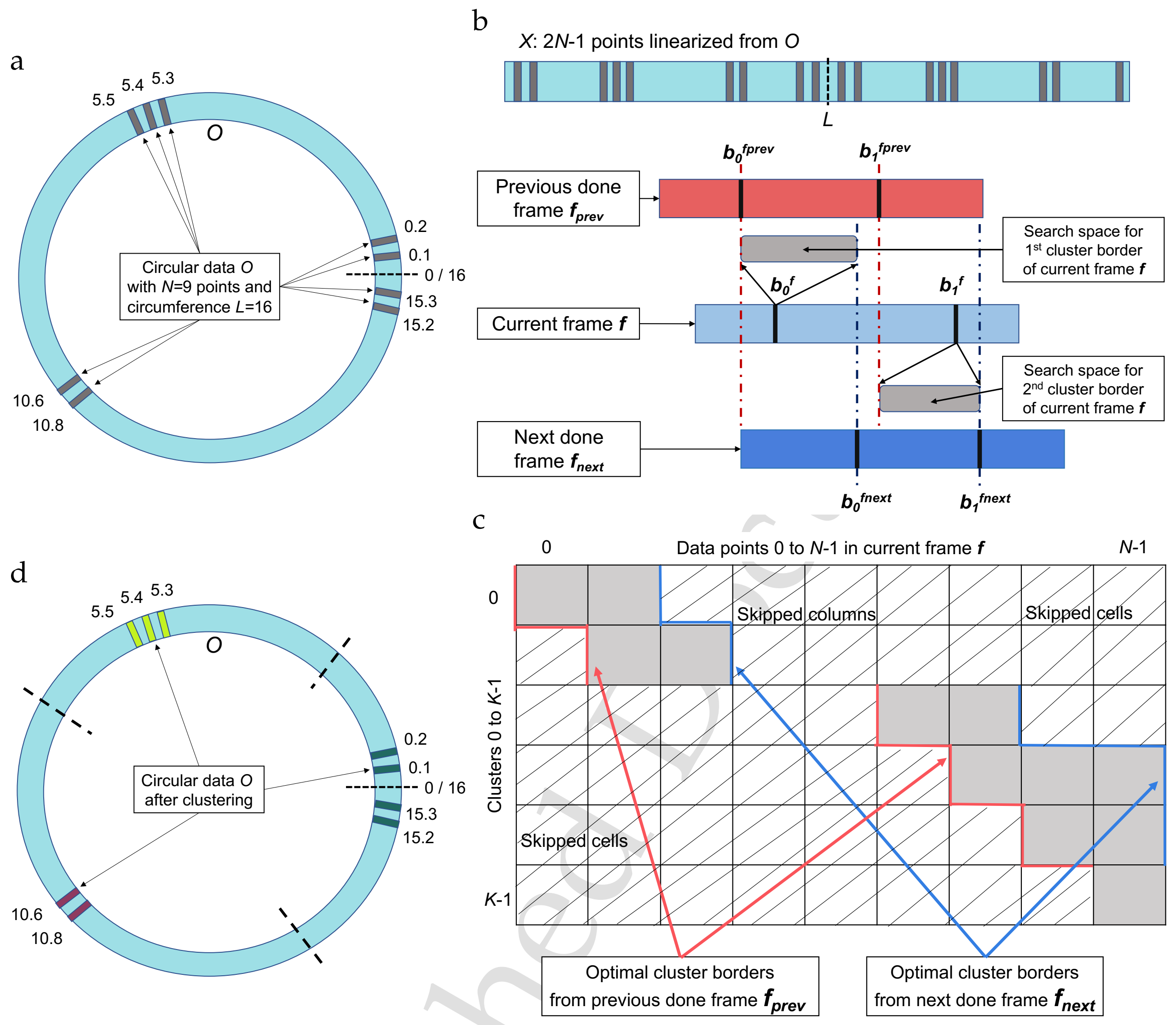

a
5.5
5.4
5.3
O
0.2
0.1
0 / 16
15.3
15.2
10.6
10.8
Circular data O with N=9 points and circumference L=16
b
X: 2N-1 points linearized from O
L
b0fprev
b1fprev
Previous done frame fprev
Search space for 1st cluster border of current frame f
b0f
b1f
Current frame f
Search space for 2nd cluster border of current frame f
Next done frame fnext
b0fnext
b1fnext
c
0
Data points 0 to N-1 in current frame f
N-1
Clusters 0 to K-1
K-1
Skipped columns
Skipped cells
Skipped cells
Optimal cluster borders from previous done frame fprev
Optimal cluster borders from next done frame fnext
d
Circular data O after clustering

Figure 1: **An overview of the fast optimal circular clustering algorithm. (a)** The input circular data O increase counterclockwise from the origin (dashed line) on a circle of circumference L. The gray line segments represent events located on the circle. **(b)** Optimal framed clustering on X linearized from O. It uses divide-and-conquer twice to reduce the search space in dynamic programming. The cluster borders of current frame f is constrained to be within borders of already done previous and next frames. **(c)** Search space reduction in the dynamic programming matrix that contains the within-cluster sum of squared distances for sub-problems in the current frame. Only gray entries in the matrix are computed. **(d)** An output optimal clustering for the input circular data O. Colors of line segments indicate three optimal clusters found.

sively reduces the search space of cluster borders.

- The FOCC algorithm guarantees optimal clusters on circular data.
- The FOCC algorithm runs in subquadratic time, much faster than other known options.

The rest of this article is structured as follows. Section 2 discusses related work and highlights relevant algorithms. Section 3 presents the FOCC algorithm and proves its correctness via the property of monotonically increasing cluster borders across frames. It also derives the asymptotic runtime of the algorithm. Section 4 compares the performance of the FOCC algorithm with existing methods and presents results on simulated and real data. Section 5 discusses applications of the algorithm to angular, periodic, and looped data clustering. Finally, Section 6 concludes the work.

2 Related Work

In contrast to clustering in the Cartesian coordinate system, circular clustering methods [22], [23], [24], [25], [26] have been sparsely designed to group circular data [27]. The MSBC algorithm [22] is a mean-shift based method. MSBC requires the user to select some parameters and transforms circular clustering to hierarchical clustering. An expectation maximization based parametric method was introduced in [23] and [24]. Two methods named SWGMM [25] and JCLMM [26] cluster circular–linear data in a cylindrical coordinate system. Both methods are based on expectation maximization and use mixture models to cluster the circular-linear data. Hierarchical clustering and expectation maximization are heuristic methods; they can converge to a locally optimal solution which may not guarantee a globally optimal clustering.

Linear clustering methods like heuristic K-means [18], fuzzy c-means [28], and dynamic programming [20], [21] algorithms can be adapted to cluster linearized circular data by considering all starting positions. However, such strategies are inefficient for large circular datasets [5]. We refer to the circular application of heuristic K-means as heuristic circular clustering (HEUC) and that of dynamic programming as brute-force circular clustering (BOCC).

The HEUC algorithm does not guarantee clustering optimality. On the other hand, the BOCC algorithm finds one set of optimal cluster borders but in quadratic time $O(KN^2)$ for circular data. Therefore, a faster algorithm to find optimal cluster borders is highly desirable. This paper proposes such an efficient algorithm called FOCC for circular clustering in subquadratic time. The FOCC algorithm takes advantage of an inherent property of the circular clustering problem to guarantee much reduced runtime and optimality.

3 METHODS

The circular clustering problem is to identify K groups from input circular data such that the total within-cluster sum of squared distances is minimized. We present the FOCC algorithm to solve the problem with guaranteed optimality, linear-polylogarithmic time complexity, and reproducibility. We will first describe the FOCC algorithm and its supporting algorithms. Then we prove its optimality based on the property of monotonically increasing cluster borders across frames. Lastly, we establish the worst-case asymptotic runtime of the FOCC algorithm.

3.1 Notation

The input to the FOCC algorithm is the circular data O, the number of clusters K, and the circumference L of the circle where data are located. We summarize symbols that are used in algorithms and proofs to be presented as follows:

O:	an unsorted array of length N to hold the coordinates along the circle for the input circular data points
L:	the circumference of the circle where data are located
K:	the number of clusters to be found on the circular data
X:	a sorted array of length $2N-1$ holding linearized data from O. X is created by appending the sorted circular data O to itself, but excluding the last point.
f:	the ID of a frame of fixed length N. ID is the index of the frame's first point in X. The range of values for f is $[-1, 0, \ldots, N-1, N]$ including two sentinel frames on both ends for boundary conditions
f_{start}:	the ID of the first frame to be clustered among a consecutive number of frames
f_{end}:	the ID of the last frame to be clustered among a consecutive number of frames
f_{prev}:	the ID of a nearest frame previously clustered smaller than the current frame ID
f_{next}:	the ID of a nearest frame previously clustered greater than the current frame ID
$\mathcal{C}^f$:	$\mathcal{C}^f = \{b_0^f, b_1^f, \ldots, b_{K-1}^f\}$ is the ending indices of each cluster in an optimal K-clustering of frame f on linearized data X. We also define a sentinel value $b_{-1}^f = f-1$ as the index to the point before the first data point in cluster 0 in frame f. This value is implicitly used in Eq. (1)
SSQ:	$SSQ(X, \mathcal{C}^f)$ is the total within-cluster sum of squared distances for a K-clustering of frame f on linearized data X

3.2 The FOCC Algorithm

The input data are coordinates on a circle in the range of $[0, L)$. For any input data point outside this range, they can be adjusted by performing a modulo L operation on that data point.

Algorithm 1 Fast-Optimal-Circular-Clustering (FOCC) encapsulates a hybrid of two divide-and-conquer algorithms and one dynamic programming algorithm. It has three main steps. The first step is to sort, linearize, and extend the circular data. The original circular locations O form the first half of linearized points. The second half is obtained by shifting the original circular data by circumference L by $O+L$, excluding the last point in O. The two sorted halves together constitute $2N-1$ linearized points in X.

In the second step, the FOCC algorithm treats the linearized data as N overlapping frames each containing N points. The starting locations of the N frames are from $X[0]$ to $X[N-1]$. A frame is numbered by the index of its first point in X. Two sentinel frames of indices -1 and N are created to handle boundary conditions. It calls Alg. 2 Framed-Clustering (FC) to perform optimal univariate linear clustering on all the frames and identifies one frame with the minimum within-cluster sum of squared distances (SSQ). Framed-Clustering identifies a frame f to minimize SSQ defined on a clustering $\mathcal{C}^f$ of the frame by

$$SSQ(X, \mathcal{C}^f) = \sum_{k=0}^{K-1} \sum_{i=b_{k-1}^f+1}^{b_k^f} (x_i - \mu_k^f)^2 \tag{1}$$

where μ_k^f is the mean of points in cluster k in clustering $\mathcal{C}^f$.

In the third step, the FOCC algorithm assigns the optimal cluster borders obtained from the FC algorithm to the original data O. The final output of FOCC is an optimal cluster assignment A to all points in O.

Algorithm 1 Fast-Optimal-Circular-Clustering(O, K, L)

```
 1: Step 1. Sort, linearize, extend circular data to linear data:
 2: Sort: Let $I$ be a permutation of $0, \dots, N-1$ such that
    $O[I[i]] \le O[I[i+1]]$ for $i = 0, \dots, N-2$
 3: Linearize: $X \leftarrow (O[I[0]], \dots, O[I[N-1]])$
 4: Extend: $X \leftarrow (X, X[0]+L, \dots, X[N-2]+L)$
 5: Step 2. Find one of $N$ frames 0 to $N-1$ to minimize within-
    cluster sum of squared distances:
 6: Frame $-1$ clustering: $\mathcal{C}^{-1} = \{b_0^{-1} = \cdots = b_{K-1}^{-1} = 0\}$
 7: Frame $N$ clustering: $\mathcal{C}^{N} = \{b_0^{N} = \cdots = b_{K-1}^{N} = 2N-1\}$
 8: $\mathcal{C}^{f^*}$, $f^* \leftarrow$ FC($X$, $K$, 0, $N-1$, $-1$, $N$)
 9: Step 3. Assign clusters to points in original circular data $O$:
10: $k \leftarrow 0$
11: for $i \leftarrow f^*$ to $f^* + N - 1$ do
12:     if $i > b_k^{f^*}$ from $\mathcal{C}^{f^*}$ then
13:         $k \leftarrow k + 1$
14:     end if
15:     $A[I[i \mod N]] \leftarrow k$
16: end for
17: return Cluster assignment $A$ in order of each point in $O$
```

The key innovation in our solution is Alg. 2 Framed-Clustering (FC) that solves the optimal framed clustering problem based on divide-and-conquer. This algorithm divides each problem into three sub-problems: namely, left, right sub-problems and the middle frame. First the middle frame is solved by calling Alg. 3 Bracket-Dynamic-Programming (BDP). The left/right sub-problem is to find an optimal frame among all frames to the left/right of the middle frame. Both sub-problems are recursively solved. The FC algorithm finally returns one optimal frame f^* and its optimal clustering $\mathcal{C}^{f^*}$.

Algorithm 2 Framed-Clustering FC(X, K, f_{start}, f_{end}, f_{prev}, f_{next})

```
 1: $f^*, \mathcal{C}^{f^*} \leftarrow$ NIL
 2: if $f_{\text{start}} \le f_{\text{end}}$ then
 3:     $f_{\text{mid}} \leftarrow \lfloor (f_{\text{start}} + f_{\text{end}})/2 \rfloor$
 4:     $\mathcal{C}^{f_{\text{mid}}} \leftarrow$BDP($X$, $K$, $f_{\text{mid}}$, $f_{\text{prev}}$, $f_{\text{next}}$)
 5:     $f^* \leftarrow f_{\text{mid}}$
 6:     $\mathcal{C}^{f^*} \leftarrow \mathcal{C}^{f_{\text{mid}}}$
 7:     $\mathcal{C}^{f_{\text{left}}}$, $f_{\text{left}} \leftarrow$ FC($X$, $K$, $f_{\text{start}}$, $f_{\text{mid}} - 1$, $f_{\text{prev}}$, $f_{\text{mid}}$)
 8:     if $SSQ(X, \mathcal{C}^{f_{\text{left}}}) \le SSQ(X, \mathcal{C}^{f^*})$ then
 9:         $f^* \leftarrow f_{\text{left}}$
10:         $\mathcal{C}^{f^*} \leftarrow \mathcal{C}^{f_{\text{left}}}$
11:     end if
12:     $\mathcal{C}^{f_{\text{right}}}$, $f_{\text{right}} \leftarrow$ FC($X$, $K$, $f_{\text{mid}} + 1$, $f_{\text{end}}$, $f_{\text{mid}}$, $f_{\text{next}}$)
13:     if $SSQ(X, \mathcal{C}^{f_{\text{right}}}) < SSQ(X, \mathcal{C}^{f^*})$ then
14:         $f^* \leftarrow f_{\text{right}}$
15:         $\mathcal{C}^{f^*} \leftarrow \mathcal{C}^{f_{\text{right}}}$
16:     end if
17: end if
18: return $\mathcal{C}^{f^*}, f^*$
```

Algorithm 3 BDP performs dynamic programming on a frame by utilizing a search bracket bounded by the optimal cluster borders already computed for two nearby enclosing frames f_{prev} and f_{next}. It executes divide-and-conquer on the data points belonging to the search bracket to fill up two $K \times N$ dynamic programming matrices S and J of the current frame f. $S[k, i]$ is the minimum SSQ value if $X[f]$ to $X[f+i]$ are put into $k+1$ optimal clusters (numbered 0 to k). $f + J[k, i]$ is the index to the first point in cluster k. The dynamic programming recurrence equations are

$$S[k,i] = \begin{cases} +\infty & i < k-1 \\ ssq(0,i,f) & k = 0 \\ \min\limits_{k-1 \le j \le i} S[k-1, j-1] + ssq(j,i,f) & \text{otherwise} \end{cases} \quad (2)$$

$$J[k,i] = \begin{cases} \text{undefined} & i < k-1 \\ 0 & k = 0 \\ \operatorname*{argmin}\limits_{k-1 \le j \le i} S[k-1, j-1] + ssq(j,i,f) & \text{otherwise} \end{cases} \quad (3)$$

where $ssq(j, i, f)$ computes the sum of squared distances from each point of $X[j+f]$ to $X[i+f]$ to the mean of the same points. Algorithm 3 BDP fills up matrices S and J only partially specified by the brackets starting within $[j_{\min}, j_{\max}]$ and ending within $[i_{\min}, i_{\max}]$ that constrain the beginning position of cluster k. It calls Alg. 4 Find-Borders, the second divide-and-conquer algorithm, to accomplish fast calculation inside the brackets. The BDP algorithm yields optimal cluster borders for the current frame.

Algorithm 3 Bracket-Dynamic-Programming BDP(X, K, f, f_{prev}, f_{next})

```
1: for $k \leftarrow 0$ to $K - 1$ do
2:     $i_{\min} \leftarrow b_k^{f_{\text{prev}}} - f_{\text{prev}}$; $i_{\max} \leftarrow b_k^{f_{\text{next}}} - f_{\text{next}}$
3:     $j_{\min} \leftarrow b_{k-1}^{f_{\text{prev}}} - f_{\text{prev}} + 1$; $j_{\max} \leftarrow b_{k-1}^{f_{\text{next}}} - f_{\text{next}} + 1$
4:     Search for cluster $k$ borders of frame $f$ starting at $j \in$
       $[j_{\min}, j_{\max}]$ and ending at $i \in [i_{\min}, i_{\max}]$:
       Find-Borders($X$, $K$, $f$, $k$, $i_{\min}$, $i_{\max}$, $j_{\min}$, $j_{\max}$, $S$, $J$)
5: end for
6: $\mathcal{C}^f \leftarrow$ Backtrack($J$, $N$, $K$, $f$)
7: return $\mathcal{C}^f$
```

Algorithm 4 Find-Borders uses divide-and-conquer to compute entries in the dynamic programming matrices bounded by given brackets of both the positions needed to be filled and the cluster borders. This is analogous to the Fill-Row algorithm defined on page N5-8 of Supplementary Note N5 [21]. Algorithm 4 adapts the Fill-Row algorithm to framed clustering.

Algorithm 4 Find-Borders(X, K, f, k, $i_{\min}$, $i_{\max}$, $j_{\min}$, $j_{\max}$, S, J)

```
1: if $i_{\min} \le i_{\max}$ then
2:     $i = \lfloor (i_{\max} + i_{\min})/2 \rfloor$
3:     Minimize-SSQ($X$, $K$, $f$, $k$, $i$, $j_{\min}$, $j_{\max}$, $S$, $J$ )
4:     Find-Borders($X$, $K$, $f$, $k$, $i_{\min}$, $i - 1$, $j_{\min}$, $J[k, i]$, $S$, $J$)
5:     Find-Borders($X$, $K$, $f$, $k$, $i + 1$, $i_{\max}$, $J[k, i]$, $j_{\max}$, $S$, $J$)
6: end if
7: return Updated entries in $S$ and $J$
```

Algorithm 5 Minimize-SSQ computes one entry at $[k, i]$ of the dynamic programming matrices S and J for a k-clustering of frame f ending at $X[i]$ of index i. Matrix S contains the SSQ values and J contains optimal cluster borders for sub-problems in the given frame. The Minimize-SSQ algorithm invokes the recurrence equations of dynamic

programming. Algorithm 5 adapts the FindMinimum algorithm defined on page N5-8 of Supplementary Note N5 [21] to framed clustering.

Algorithm 5 Minimize-SSQ(X, K, f, k, i, $j_{\min}$, $j_{\max}$, S, J)

```
 1: if $i < k$ then
 2:     $S[k, i] = \infty$
 3: else if $k = 0$ or $i = 0$ then
 4:     $S[k, i] = ssq(0, i, f)$
 5:     $J[k, i] = 0$
 6: else
 7:     $S[k, i] = \infty$
 8:     $J[k, i] = i$
 9:     for $j = \max(j_{\min}, k)$ to $\min(j_{\max}, i)$ do
10:         if $S[k-1, j-1] + ssq(j, i, f) \le S[k, i]$ then
11:             $S[k, i] = S[k-1, j-1] + ssq(j, i, f)$
12:             $J[k, i] = j$
13:         end if
14:     end for
15: end if
16: return Updated entries in $S$ and $J$
```

Algorithm 5 calls function $ssq(j, i, f)$ $(j \le i)$, whose calculation takes $O(j - i + 1)$ time by definition. However, with pre-computed sums and sums of squares, it can be done in constant time $O(1)$, critical for the overall runtime to stay below quadratic in N. The pre-computed sums are

$$Z[i] = \sum_{l=0}^{i} X[l] \quad i = 0, \ldots, 2N-1 \tag{4}$$

and the pre-computed sums of squares are

$$Q[i] = \sum_{l=0}^{i} X^2[l] \quad i = 0, \ldots, 2N-1 \tag{5}$$

The mean of $X[j+f]$ to $X[i+f]$ is computed in constant time by

$$\mu(j, i, f) = \begin{cases} \frac{Z[i]}{i+1} & j = f = 0 \\ \frac{Z[i+f] - Z[j+f-1]}{i-j+1} & \text{otherwise} \end{cases} \quad (0 \le j \le i) \tag{6}$$

and finally, $ssq(j, i, f)$ is also computed in constant time by

$$ssq(j, i, f) = Q[i] - \frac{1}{i+1} Z^2[i], \quad j = f = 0 \tag{7}$$

or, when $j = f = 0$ is not true,

$$ssq(j, i, f) = Q[i+f] - Q[j+f-1] - (i - j + 1)\mu^2(j, i, f) \tag{8}$$

Supplementary Note N5 [21] derived these equations in full detail. We replace i and j there with $i+f$ and $j+f$ here, respectively, to incorporate the frame concept. Additionally, as framed clustering is unweighted, we also replace weights by one from equations there [21].

Algorithm 6 Backtrack retrieves an optimal K-clustering borders for frame f in linear time of K from matrix J. It adapts the Backtrack algorithm given on page N5-3 of Supplementary Note N5 [21] to framed clustering.

Algorithm 6 Backtrack(J, N, K, f)

```
 1: Initialize $b$ to hold ending indices of $K$ clusters
 2: $j = N - 1$
 3: $k = K - 1$
 4: $b[k] = j + f$
 5: while $q > 0$ do
 6:     $j = J[k, i] - 1$
 7:     $k = k - 1$
 8:     $b[k] = j + f$
 9: end while
10: return $C^f = \{b[0], \ldots, b[K-1]\}$
```

3.3 The correctness and optimality of FOCC

Here, we establish the correctness and optimality of the FOCC algorithm. We will show that the use of bracket dynamic programming, while reducing the runtime, always guarantees an optimal clustering solution. The proof is based on the monotonically increasing property of cluster borders across frames when SSQ is to be minimized. By monotonically increasing, we mean that we can always find optimal cluster borders of a current frame to be greater than or equal to those corresponding optimal borders in a previous frame that starts before the current frame. This property ensures that some optimal solutions can always be found within the bracket formed by two neighboring frames enclosing a current frame.

Lemma 1 (Monotonically increasing cluster borders)**.** *Let $x_0 \le \cdots \le x_i$ be a sorted sequence of $i+1$ numbers. Let $j_k(i-1)$ be the beginning index of cluster k ending at index $i-1$ in an optimal $(k+1)$-clustering of the first i numbers. Let $j_k(i)$ be the beginning index of cluster k ending at i in an optimal $(k+1)$-clustering of all $i+1$ numbers. Given $j_k(i-1)$ or $j_k(i)$, we can always find the other such that*

$$j_k(i) \ge j_k(i-1) \tag{9}$$

This is a well-known monotonic property of the univariate clustering problem that minimizes the within-cluster sum of squared distances [21]. One proof is provided in Supplementary Note N5 [21] as Theorem N5.4.4 (page N5-6), which was stated for weighted optimal clustering whose borders for each cluster are the largest possible when there are multiple optimal clustering solutions. Evidently, the proof applies to unweighted optimal clustering. Lemma 1 rephrased the theorem for the context needed here.

Lemma 2. *Let $x_0 \le \cdots \le x_i$ be a sorted sequence of $i+1$ numbers. Let $b_q(0, i-1)$ be the ending index of cluster q in an optimal $(k+1)$-clustering of the first i numbers. Let $b_q(0, i)$ be the ending index of cluster q in an optimal $(k+1)$-clustering of all $i+1$ numbers. Given $b_q(0, i)$ or $b_q(0, i-1)$, we can always find the other to satisfy*

$$b_q(0, i) \ge b_q(0, i-1), \quad q = 0, \ldots, k \tag{10}$$

Proof. (By induction)

Base case: By definition and when $q = k$, we have $b_k(0, i) = i$ and $b_k(0, i-1) = i-1$, so we have $b_k(0, i) \ge b_k(0, i-1)$.

Hypothesis: $b_q(0, i) \ge b_q(0, i-1)$ for cluster q or higher.

Induction: Let $j_q(h)$ be the beginning index of cluster q ending at index h. For cluster $q-1$ of the clustering on all i points, its ending index is $b_{q-1}(0,i) = j_q(b_q(0,i)) - 1$; for cluster $q-1$ of the clustering on x_0 to x_{i-1}, its ending index is $b_{q-1}(0,i-1) = j_q(b_q(0,i-1)) - 1$. By the induction hypothesis, we have $b_q(0,i) \geq b_q(0,i-1)$. Applying Lemma 1, we can find $j_q(b_q(0,i)) \geq j_q(b_q(0,i-1))$. It leads to $b_{q-1}(0,i) \geq b_{q-1}(0,i-1)$, proving the induction hypothesis. □

Lemma 3. *Let $x_0 \leq \cdots \leq x_i$ be a sorted sequence of $i+1$ numbers. Let $b_q(1,i)$ be the ending index of cluster q in an optimal $(k+1)$-clustering of the last i numbers. Let $b_q(0,i)$ be the ending index of cluster q in an optimal $(k+1)$-clustering of all $i+1$ numbers. Given $b_q(0,i)$, we can always find $b_q(1,i)$ such that*

$$b_q(1,i) \geq b_q(0,i), \quad q = 0, \ldots, k \tag{11}$$

Proof. Reflecting $x_0, \ldots, x_i$ around zero, we obtain $x'_0 = -x_i \leq x'_1 = -x_{i-1} \leq \cdots \leq x'_i = -x_0$. As distances between points are preserved, an optimal $(k+1)$-clustering on x'_0 to x'_i also maps to an optimal $(k+1)$-clustering on x_0 to x_i. It follows that

$$b'_{k-q-1}(0,i) = i - b_q(0,i) - 1, \quad q = 0, \ldots, k-1 \tag{12}$$

With clustering on x_1 to x_i mapping to clustering on x'_0 to x'_{i-1}, we have

$$b'_{k-q-1}(0,i-1) = i - b_q(1,i) - 1, \quad q = 0, \ldots, k-1 \tag{13}$$

By Lemma 2, we can find $b'_{k-q-1}(0,i-1) \leq b'_{k-q-1}(0,i)$. From inequalities (12) and (13), we immediately have

$$b_q(1,i) \geq b_q(0,i), \quad q = 0, \ldots, k-1 \tag{14}$$

When $q = k$, $b_k(0,i) = x_i = b_k(1,i)$. Therefore, we have

$$b_q(1,i) \geq b_q(0,i), \quad q = 0, \ldots, k \tag{15}$$

which proves the claim in this lemma. □

Lemma 4. *Let f be the starting index of a frame. Let $(b_0^f, \ldots, b_{K-1}^f)$ be the ending index of each cluster in an optimal K-clustering of points within frame f. Then there must exist an optimal K-clustering of frame $f+1$ such that*

$$b_k^f \leq b_k^{f+1}, \qquad 0 \leq k \leq K-1 \tag{16}$$

Proof. By definition, frame f contains points $X[f], \ldots, X[f+N]$ and frame $f+1$ contains $X[f+1], \ldots, X[f+N+1]$. We create an intermediate subset $X[f], \ldots, X[f+N+1]$. By Lemma 2, we have $b_k(f, f+N) \leq b_k(f, f+N+1)$ on K-clustering of frame f and the intermediate subset of X; by Lemma 3, we have $b_k(f, f+N+1) \leq b_k(f+1, f+N+1)$ on the intermediate subset of X and frame $f+1$. Integrating the two inequalities, we obtain

$$b_k(f, f+N) \leq b_k(f+1, f+N+1), \; 0 \leq k \leq K-1 \tag{17}$$

By definition, $b_k(f, f+N) = b_k^f$ and $b_k(f+1, f+N+1) = b_k^{f+1}$. Therefore, we have

$$b_k^f \leq b_k^{f+1}, \qquad 0 \leq k \leq K-1 \tag{18}$$

which proves the lemma. □

Theorem 1 (Monotonically increasing cluster borders across frames)**.** *Let f be the starting index of a frame. Let $(b_0^f, \ldots, b_{K-1}^f)$ be the ending index of each cluster in an optimal K-clustering of points within frame f. Then there must exist an optimal K-clustering of frame f whose cluster borders indices are bounded between any previous frame $f_{prev} < f$ and any next frame $f_{next} > f$, that is*

$$b_k^{f_{prev}} \leq b_k^f \leq b_k^{f_{next}}, \qquad 0 \leq k \leq K-1 \tag{19}$$

Proof. As $f_{\text{prev}} < f$, we apply Lemma 4 repeatedly on consecutive pairs of frames starting at f_{prev} and ending at f to get

$$b_k^{f_{\text{prev}}} \leq b_k^{f_{\text{prev}}+1} \leq \cdots \leq b_k^{f-1} \leq b_k^f, \qquad 0 \leq k \leq K-1 \tag{20}$$

which leads to

$$b_k^{f_{\text{prev}}} \leq b_k^f, \qquad 0 \leq k \leq K-1 \tag{21}$$

As $f < f_{\text{next}}$, by applying Lemma 4 repeatedly on consecutive pairs of frames from f to f_{next}, we can similarly derive

$$b_k^f \leq b_k^{f_{\text{next}}}, \qquad 0 \leq k \leq K-1 \tag{22}$$

Therefore, we can conclude there must exist optimal K-clustering of frame f such that its cluster ending indices are bounded by those of frames f_{prev} and f_{next}, satisfying

$$b_k^{f_{\text{prev}}} \leq b_k^f \leq b_k^{f_{\text{next}}}, \qquad 0 \leq k \leq K-1 \tag{23}$$

which proves the theorem. □

Theorem 2. *The FOCC correctly returns a K-clustering of the input circular data that minimizes the within-cluster sum of squared distances.*

Proof. With K clusters, there are exactly K cluster borders to be determined on circular data. Once the beginning position of the first cluster is given, the circular clustering problem reduces to a linear clustering problem. As there are N possible start positions of the first cluster, the circular clustering problem needs to solve N linear clustering sub-problems. Algorithm 1 FOCC indeed solves exactly these sub-problems. Next we justify the correctness of FOCC's constituent algorithms.

Algorithm 2 Framed-Clustering uses divide-and-conquer to find a frame with the minimum SSQ. It indeed covers each frame exactly once by bracket dynamic programming. Remaining unprocessed frames are passed onto next level of recursion.

Algorithm 3 Bracket-Dynamic-Programming correctly solves linear univariate clustering without bracketing [20], [21]. As Theorem 1 guarantees that a set of optimal borders must belong to brackets formed by two neighboring frames already computed, optimal solutions to sub-problems must be found by searching for optimal borders within each bracket during dynamic programming.

Finally, Alg. 1 FOCC uses cluster borders of the optimal frame to assign clusters to each point in the original circular data, providing the correct solution to the original circular clustering problem. □

3.4 The asymptotic runtime of FOCC

In addition to the speedup due to bracket dynamic programming by Alg. 3 BDP, the divide-and-conquer in Alg. 2 FC processes frames in pre-order on a binary tree of all frames, instead of in the order of frame positions along X. This strategy maximizes time savings due to bracket dynamic programming. The runtime of solving a single frame by Alg. 4 is based on integrating the brackets into a log-linear solution for univariate linear clustering previously established [21].

Theorem 3. *The worst-case asymptotic runtime of the FOCC algorithm is $\mathcal{O}(KN\log^2 N)$, where N is the number of circular data points and K is the number of clusters.*

Proof. We first establish the runtime for a given $k \in [0, K-1]$. In Alg. 3 BDP, for frame f only the bracket $[i_{\min}, i_{\max}]$ is computed for row k in S and J matrices, where the optimal cluster boundaries are searched for within the bracket $[j_{\min}, j_{\max}]$. Let $m_i(f) = i_{\max} - i_{\min} + 1$ and $m_j(f) = j_{\max} - j_{\min} + 1$. We know from previous univariate clustering results [21] that it takes $O(m_j(f)\log m_i(f))$ time to fill out the elements in $S[k, i_{\min}]$, ..., $S[k, i_{\max}]$ and $J[k, i_{\min}]$, ..., $J[k, i_{\max}]$.

At the recursion depth $d \in [0, \lfloor \log N \rfloor]$ of Alg. 2 Framed-Clustering, exactly 2^d frames are computed. Let these frames be $f_{p_1}, \ldots, f_{p_{2^d}}$. The time $H(k, d, N)$ to compute brackets within these frames at depth d is thus

$$H(k, d, N) = \sum_{r=1}^{2^d} m_j(f_{p_r}) \log m_i(f_{p_r}) \tag{24}$$

As these ~~frames~~ brackets overlap by exactly one boundary element, it must follow that

$$\sum_{r=1}^{2^d} m_j(f_{p_r}) = 2N - 1 + 2^d - 1 \tag{25}$$

$$\sum_{r=1}^{2^d} m_i(f_{p_r}) = 2N - 1 + 2^d - 1 \tag{26}$$

Replacing $m_i(f_{p_r})$ in Eq. (24) by a larger value of $\sum_{s=1}^{2^d} m_i(f_{p_s})$, we derive an upper bound for $H(k, d, N)$:

$$\begin{aligned}
&H(k, d, N) && (27)\\
&\le \sum_{r=1}^{2^d} m_j(f_{p_r}) \log \sum_{s=1}^{2^d} m_i(f_{p_s}) && (28)\\
&= (2N - 1 + 2^d - 1)\log(2N - 1 + 2^d - 1) && (29)\\
&\le (2N - 1 + 2N - 1)\log(2N - 1 + 2N - 1) && (30)\\
&\le 4N \log 4N && (31)
\end{aligned}$$

Summing up $H(k, d, N)$over depth d and k, we have an upper bound to the runtime for Alg. 2 Framed-Clustering:

$$\sum_{k=0}^{K-1} \sum_{d=0}^{\lfloor \log N \rfloor} H(k, d, N) \le 4KN \log^2 4N \tag{32}$$

which dominates the $\mathcal{O}(N \log N)$ time for sorting the circular data in step 1 and the linear time for cluster assignment in step 3 of Alg. 1 FOCC. Therefore, the overall runtime $T(N, K)$ of FOCC in the worst case is asymptotically $T(N, K) = \mathcal{O}(KN \log^2 N)$. □

4 Results

We now evaluate the performance of the FOCC algorithm in contrast to HEUC and BOCC algorithms on simulated circular data and real circular data from three round genomes. We report the observed runtime of each algorithm as a function of sample size and number of clusters, and the clustering accuracy measured in within-cluster sum of squared distances. We also illustrate qualitative differences of the clusters produced on both real and simulated data by optimal and heuristic clustering.

4.1 Optimality and runtime on simulated data

We simulated circular data to evaluate the runtime, accuracy, and cluster quality of FOCC, BOCC, and HEUC. Linear data were randomly generated from Gaussian mixture models where each Gaussian component represents a cluster. Linear data are converted to circular data by the modulo operation.

For the first experiment with results shown in Figure 2(a) and (c), we created a Gaussian mixture model comprising of three components. Each component had 500 random data points, which modulo the circumference 210 of the circle are used as the input O. Their means were $0, 100, 200$ respectively and standard deviation was 0.3 for all components.

In the second experiment with results displayed in Figure 2(b), the same Gaussian mixture model with varying sample sizes was used.

The optimality of each algorithm is visualized in Figure 2(a). The BOCC algorithm provides a gold standard as it is guaranteed to find the minimum SSQ via brute-force search of optimal clustering among all frames. The FOCC algorithm produced identical SSQ with BOCC, supporting its optimality. However, the HEUC algorithm led to SSQ values higher than the minimum SSQ when K is large, indicating that non-optimal clustering has resulted from its heuristic. This result thus confirms the theoretical argument that FOCC guarantees to find optimal circular clustering.

The runtime results are reported in Figure 2(b,c). Figure 2(b) shows that the runtime of BOCC and HEUC grows with increasing input size N polynomially faster than the runtime of FOCC. At an input size $N = 50,000$, FOCC runs about 800 times faster than HOCC and about 400 times faster than HEUC. Figure 2(c) is the runtime as a function of number of clusters K for each algorithm for fixed N. Although the runtime of each algorithm grows at a similar rate with K, the runtime of FOCC stays about 250 and 50 times lower than the BOCC and HEUC algorithms, respectively. All runtime was observed on an iMac with 2.93 GHz Intel Core i7 processor, 16 GB 1333 MHz DDR3 RAM, and a 2TB HDD. These results suggest that FOCC cashed out its theoretical advantage to be highly efficient in practice than what had been achievable for circular clustering.

Next, we examined the difference between optimal and heuristic clustering qualitatively. We created a Gaussian mixture model of three components with a standard deviation of 1 and mean $0, 5, 11$, respectively. We sampled 100

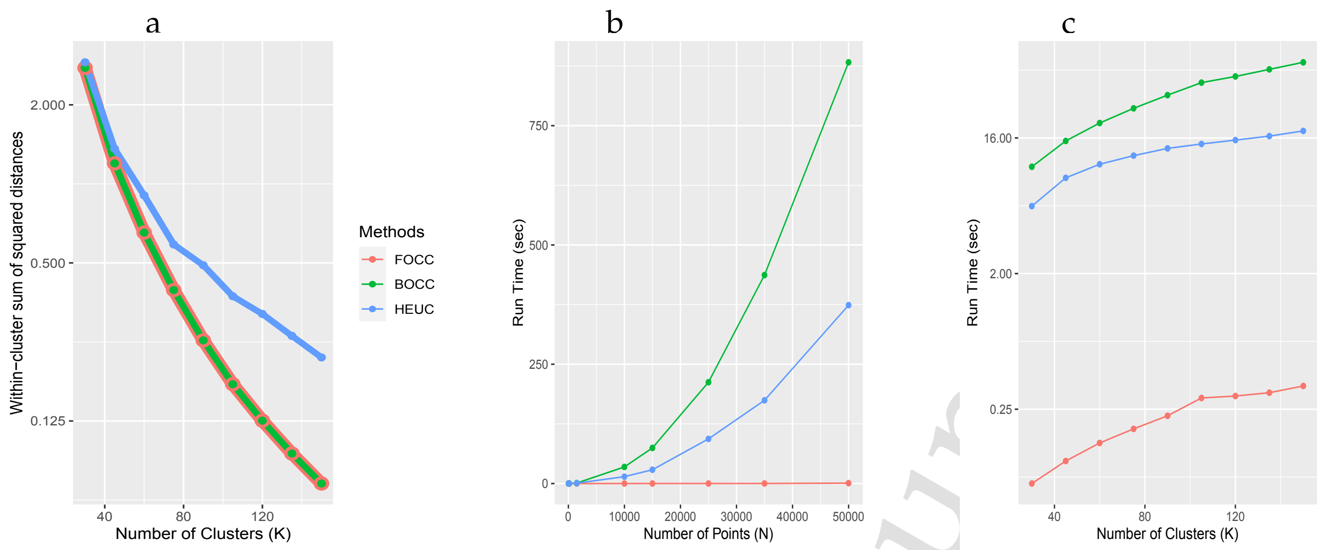
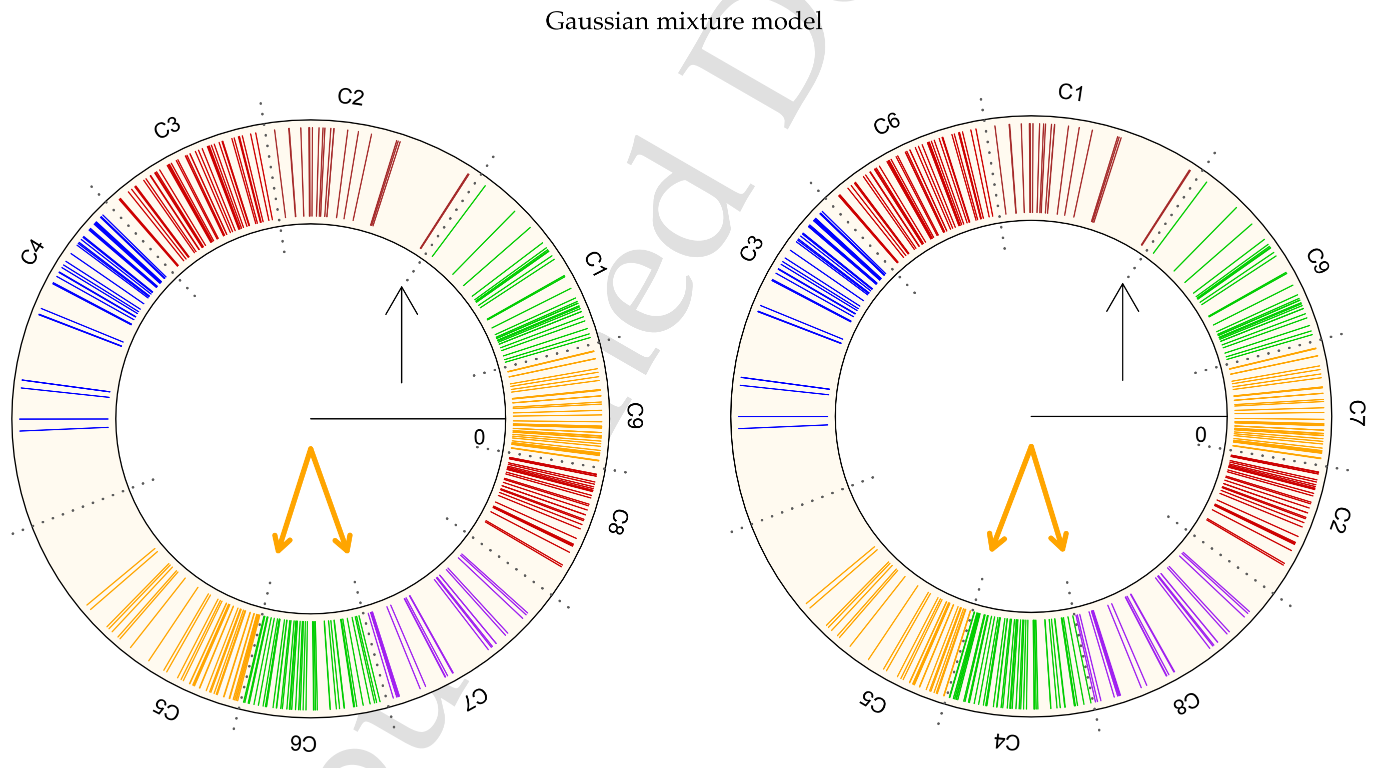


Figure 2: **The optimality and runtime of fast optimal circular clustering (FOCC) versus brute-force optimal circular clustering (BOCC) and heuristic circular clustering (HEUC). (a)** The within-cluster sum of squared distances as a function of number of clusters on the same dataset. **(b)** Runtime as a function of number of points N in the circular data for a fixed number of clusters K. **(c)** Runtime as a function of number of clusters K at a fixed sample size N.

FOCC: fast and optimal
****Recommended****

HEUC: heuristic
not always optimal



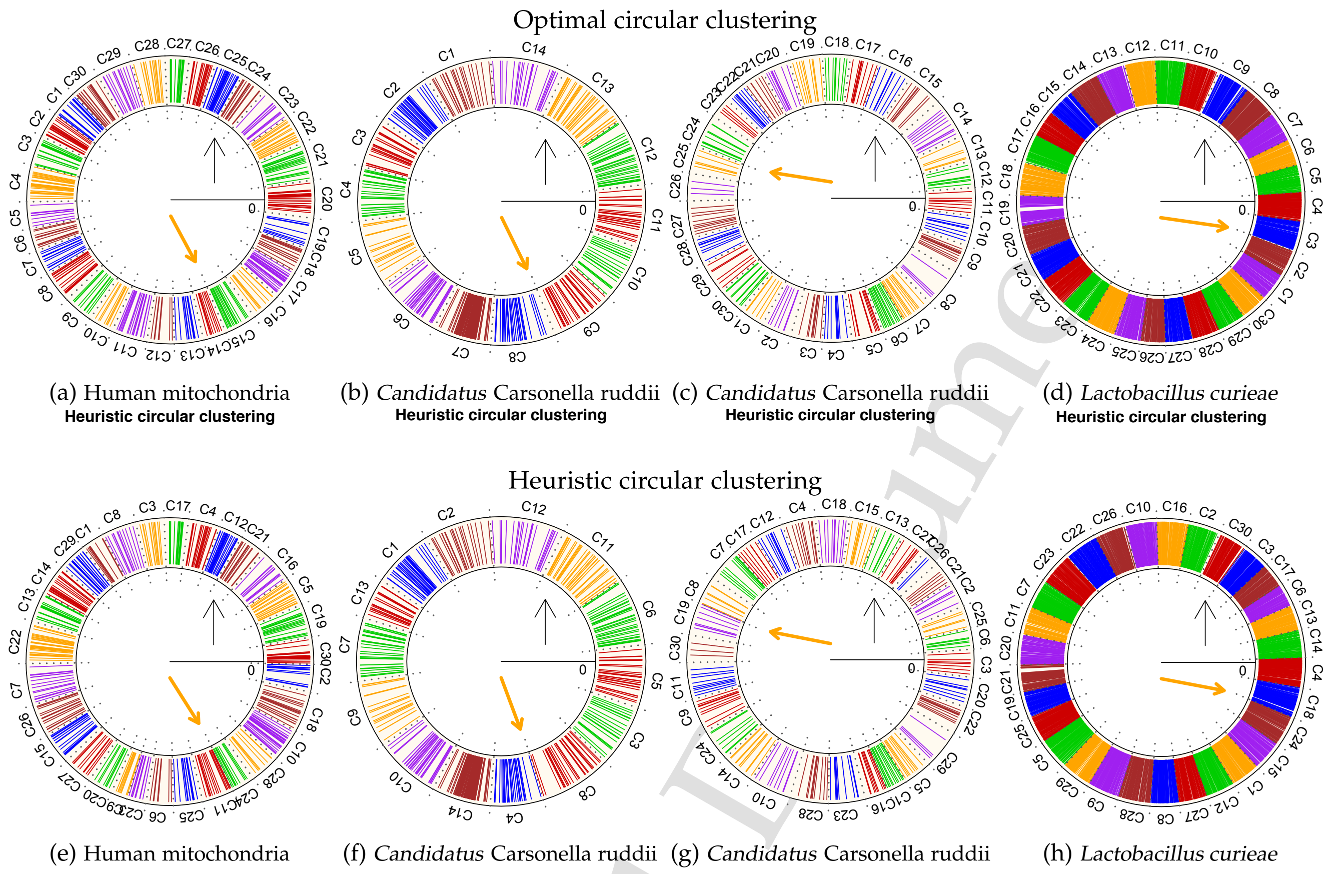
(a) Optimal clustering by FOCC

(b) Heuristic clustering by HEUC

Figure 3: **Effectiveness of optimal versus heuristic circular clustering on simulated data.** The circular data were randomly generated using a Gaussian mixture model modulo the circumference. Each solid line segment represents a circular point. The black horizontal line marks the origin of the circle. The black arrow indicates the points increasing counterclockwise. Nine optimal clusters returned by FOCC are marked in color in **(a)** and nine heuristic clusters by HEUC in **(b)**. Borders (dotted lines) between the C5 (orange) and C6 (green) clusters and between the C6 (green) and C7 (violate) clusters of the FOCC result are more justifiable as compared to the corresponding HEUC output. The FOCC algorithm puts cluster borders in wider gaps than the HEUC algorithm.

points from each component. We then mapped the points modulo $L = 15$ to a circle with circumference 15.

Figure 3 visualizes clustering outputs from FOCC and HEUC for the Gaussian mixture model. The FOCC pro-



(a) Human mitochondria (b) *Candidatus* Carsonella ruddii (c) *Candidatus* Carsonella ruddii (d) *Lactobacillus curieae*

Heuristic circular clustering Heuristic circular clustering Heuristic circular clustering Heuristic circular clustering

(e) Human mitochondria (f) *Candidatus* Carsonella ruddii (g) *Candidatus* Carsonella ruddii (h) *Lactobacillus curieae*

Figure 4: **CpG sites and gene starting sites are clustered in three round genomes captured well by optimal, but not heuristic circular clustering.** Each column used the same data. The FOCC clusters are in the top row and HEUC clusters in the bottom row. **(a)** CpG sites in 30 clusters along the human mitochondrial genome [29]. **(b)** CpG sites in 14 clusters along the *Candidatus* Carsonella ruddii genome [30]. **(c)** Gene locations in 30 clusters along the *Candidatus* Carsonella ruddii genome [31]. **(d)** Gene locations in 30 clusters along the *Lactobacillus curieae* genome [32]. **(e,f,g,h)** are clustering results using heuristic clustering on the same four datasets in (a,b,c,d), respectively.

duced justifiable clustering outcomes; however, the clusters found by HEUC are not optimal. Many cluster borders generated by the FOCC and HEUC are similar except the borders between cluster C5 and C6 and that between C6 and C7. The FOCC algorithm tends to put cluster borders in wider gaps, whereas the HEUC algorithm may identify a sub-optimal border. Thus, qualitatively it can be said that the FOCC algorithm makes better clusters than the HEUC algorithm.

Complementary to the theoretical arguments, the result on simulated data demonstrates the uncompromising practical advantages of the FOCC algorithm in both optimality and efficiency over existing algorithms for circular clustering.

4.2 Cluster quality on round genomic data

Round genomes are the most abundant among all genomes due to the large number of bacterial species. We applied circular clustering on CpG sites and gene locations from three round genomes, including the human mitochondrial genome, the *Candidatus* Carsonella ruddii genome, and the *Lactobacillus curieae* genome. The length of each genome is the circumference of data. The clustering results are shown in Figure 4.

The CpG site clustering in the human mitochondrial genome [33] is visualized in Figure 4(a,e). Mitochondria in eukaryotic cells produce ATP from food nutrients and store energy. We have extracted 30 clusters from the genome having 16,569 bp and 435 CpG sites.

Candidatus Carsonella ruddii (Ca. C. ruddii) [34] is found in phloem sap-feeding insects also known as psyllids [30]. There it synthesizes amino acids. Figure 4(b,f) shows clustering outcomes for 490 CpG sites of the Ca. C. ruddii genome. We explored 14 clusters from the 173,904 bp long genome. Similarly, Figure 4(c,g) shows gene clusters in the Ca. C. ruddii [35] genome. We have extracted 30 clusters from 232 gene locations.

The *Lactobacillus curieae* is popularly used to ferment different types of milk [36] and produce gamma-aminobutyric acid. *Lactobacillus curieae* is specially used to produce stinky tofu brine [36]. Figure 4(d,h) demonstrate gene clusters in the 2,095,860 bp long [37] genome with 2,010 genes, which are shown in 30 clusters inside the genome.

Clustering from FOCC, guaranteed to minimize within-cluster sum of squared distances, is subjectively adequate when compared to the HEUC clustering. For example, one prominent difference can be observed between Figure 4(b) and (f). The HEUC algorithm assigned sub-optimal cluster border between the C4 and C8 clusters of the CpG sites in the lower right portion of the Ca. C ruddii genome, whereas the FOCC output combines the entire compact CpG sites inside one cluster (C8). Similar examples can be found on all the other genome clusters. The FOCC output reveals underlying event patterns along these genomes. These patterns may suggest non-random biological activity along the circular genome.

5 Discussion

The FOCC algorithm can be applied to cluster angular data. Given the angular coordinate Θ (in radian) of a point in a polar coordinate system, we can convert it to a location O on a circle of circumference L by $O = L\frac{\Theta}{2\pi}$. Then we can apply the FOCC algorithm to find clusters for O which can be translated to angular clusters in the original input data.

For periodical data with period L, we can map the input data by modulo L to a circle of circumference L and then apply the FOCC algorithm to find the clusters.

The definition of circular data is not strict. In addition to be located on a circle, data points can be on a non-self-intersecting loop with a distance between two points on the loop defined as the minimum sum of distances between each consecutive pair of points along a path between the two points. The presented algorithms maintain optimality by this generalization to looped data clustering.

6 Conclusions

We have presented an algorithm for fast, reproducible, and optimal circular clustering. On both simulated and real round genomic data from mitochondria and bacteria, it outperforms in both accuracy and runtime other circular clustering methods including the heuristic K-means algorithm. We anticipate that it becomes a valuable addition to data science for the analysis of circular, periodic, angular, looped, or framed data, arising from biology and many other scientific disciplines.

Software Availability

All presented and evaluated algorithms are implemented in C/C++ and R programming languages available in an R software package 'OptCirClust' released via the Comprehensive R Archive Network. The package also includes both circular and framed data clustering visualization functions. A vignette guides the user through functions provided in the package. The package can be freely downloaded from https://CRAN.R-project.org/package=OptCirClust.

Additionally, R script files and data files are deposited to Code Ocean for reproducing figures in the results section via link https://codeocean.com/capsule/2728449/tree.

Acknowledgments

The work was partially supported by National Science Foundation grant 1661331 and USDA grant 2016-51181-25408. TD would also like to acknowledge a PhD tuition scholarship awarded to him from New Mexico State University. The authors would like to thank the anonymous reviewers for their constructive feedback towards the betterment of presentation.

References

[1] L. Landler, G. D. Ruxton, and E. P. Malkemper, "Grouped circular data in biology: advice for effectively implementing statistical procedures." *Behav Ecol Sociobiol*, vol. 74, no. 8, p. 100, 2020.

[2] L.-M. Bobay and H. Ochman, "The evolution of bacterial genome architecture," *Frontiers in Genetics*, vol. 8, p. 72, 2017. [Online]. Available: https://doi.org/10.3389/fgene.2017.00072

[3] H. Daniell, C.-S. Lin, M. Yu, and W.-J. Chang, "Chloroplast genomes: diversity, evolution, and applications in genetic engineering," *Genome Biology*, vol. 17, no. 1, p. 134, 2016. [Online]. Available: https://doi.org/10.1186/s13059-016-1004-2

[4] M. Crimi and R. Rigolio, "The mitochondrial genome, a growing interest inside an organelle." *Int J Nanomedicine*, vol. 3, no. 1, pp. 51–57, 2008.

[5] R. Amann and R. Rosselló-Móra, "After All, Only Millions?" *mBio*, vol. 7, no. 4, 2016. [Online]. Available: https://doi.org/10.1128/mBio.00999-16

[6] L. S. Kristensen, M. S. Andersen, L. V. W. Stagsted, K. K. Ebbesen, T. B. Hansen, and J. Kjems, "The biogenesis, biology and characterization of circular RNAs," *Nature Reviews Genetics*, vol. 20, no. 11, pp. 675–691, 2019. [Online]. Available: https://doi.org/10.1038/s41576-019-0158-7

[7] H. Kim, N.-P. Nguyen, K. Turner, S. Wu, A. D. Gujar, J. Luebeck, J. Liu, V. Deshpande, U. Rajkumar, S. Namburi, S. B. Amin, E. Yi, F. Menghi, J. H. Schulte, A. G. Henssen, H. Y. Chang, C. R. Beck, P. S. Mischel, V. Bafna, and R. G. W. Verhaak, "Extrachromosomal DNA is associated with oncogene amplification and poor outcome across multiple cancers," *Nature Genetics*, vol. 52, no. 9, pp. 891–897, 2020. [Online]. Available: https://doi.org/10.1038/s41588-020-0678-2

[8] S. Karlin, I. Ladunga, and B. E. Blaisdell, "Heterogeneity of genomes: measures and values." *Proc Natl Acad Sci U S A*, vol. 91, no. 26, pp. 12 837–12 841, Dec 1994.

[9] R. G. Harrison and E. L. Larson, "Heterogeneous genome divergence, differential introgression, and the origin and structure of hybrid zones." *Mol Ecol*, vol. 25, no. 11, pp. 2454–2466, Jun 2016.

[10] A. M. Deaton and A. Bird, "CpG islands and the regulation of transcription." *Genes Dev*, vol. 25, no. 10, pp. 1010–1022, May 2011.

[11] B. Foxman, "Chapter 7—Omics analyses in molecular epidemiologic studies," in *Molecular Tools and Infectious Disease Epidemiology*, B. Foxman, Ed. San Diego: Academic Press, 2012, pp. 99–116. [Online]. Available: https://doi.org/10.1016/B978-0-12-374133-2.00007-1

[12] M. Wolański, R. Donczew, A. Zawilak-Pawlik, and J. Zakrzewska-Czerwińska, "oriC-encoded instructions for the initiation of bacterial chromosome replication." *Front Microbiol*, vol. 5, p. 735, 2014.

[13] H. Sobhy, R. Kumar, J. Lewerentz, L. Lizana, and P. Stenberg, "Highly interacting regions of the human genome are enriched with enhancers and bound by DNA repair proteins," *Scientific Reports*, vol. 9, no. 1, p. 4577, 2019. [Online]. Available: https://doi.org/10.1038/s41598-019-40770-9

[14] R. Dong, L. He, R. L. He, and S. S.-T. Yau, "A novel approach to clustering genome sequences using inter-nucleotide covariance," *Frontiers in Genetics*, vol. 10, p. 234, 2019. [Online]. Available: https://doi.org/10.3389/fgene.2019.00234

[15] K. W. Govek, V. S. Yamajala, and P. G. Camara, "Clustering-independent analysis of genomic data using spectral simplicial theory." *PLoS Comput Biol*, vol. 15, no. 11, p. e1007509, Nov 2019.

[16] L. R. Mayor, K. P. Fleming, A. Müller, D. J. Balding, and M. J. E. Sternberg, "Clustering of protein domains in the human genome." *J Mol Biol*, vol. 340, no. 5, pp. 991–1004, Jul 2004.

[17] F. Dios, G. Barturen, R. Lebrón, A. Rueda, M. Hackenberg, and J. Oliver, "DNA clustering and genome complexity." *Comput Biol Chem*, vol. 53 Pt A, pp. 71–78, Dec 2014.

[18] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, March 1982.

[19] R. Bellman, "A note on cluster analysis and dynamic programming," *Mathematical Biosciences*, vol. 18, no. 3, pp. 311–312, 1973. [Online]. Available: https://doi.org/10.1016/0025-5564(73)90007-2

[20] H. Wang and M. Song, "Ckmeans.1d.dp: Optimal k-means clustering in one dimension by dynamic programming." *R J*, vol. 3, no. 2, pp. 29–33, Dec 2011. [Online]. Available: https://doi.org/10.32614/RJ-2011-015

[21] M. Song and H. Zhong, "Efficient weighted univariate clustering maps outstanding dysregulated genomic zones in

human cancers," *Bioinformatics*, vol. 36, no. 20, pp. 5027–5036, 07 2020. [Online]. Available: https://doi.org/10.1093/bioinformatics/btaa613

[22] S.-J. Chang-Chien, W.-L. Hung, and M.-S. Yang, "On mean shift-based clustering for circular data," *Soft Computing*, vol. 16, no. 6, pp. 1043–1060, 2012. [Online]. Available: https://doi.org/10.1007/s00500-012-0802-z

[23] F. Lagona and M. Picone, "A Gaussian–von Mises hidden Markov model for clustering multivariate linear-circular data," in *Statistical Models for Data Analysis*, P. Giudici, S. Ingrassia, and M. Vichi, Eds. Heidelberg: Springer International Publishing, 2013, pp. 171–179.

[24] ——, "Model-based clustering of multivariate skew data with circular components and missing values," *Journal of Applied Statistics*, vol. 39, no. 5, pp. 927–945, 2012. [Online]. Available: https://doi.org/10.1080/02664763.2011.626850

[25] A. Roy, S. K. Parui, and U. Roy, "SWGMM: a semi-wrapped Gaussian mixture model for clustering of circular–linear data," *Pattern Analysis and Applications*, vol. 19, no. 3, pp. 631–645, 2016. [Online]. Available: https://doi.org/10.1007/s10044-014-0418-2

[26] A. Roy, A. Pal, and U. Garain, "JCLMM: A finite mixture model for clustering of circular-linear data and its application to psoriatic plaque segmentation," *Pattern Recognition*, vol. 66, pp. 160–173, 2017. [Online]. Available: https://doi.org/10.1016/j.patcog.2016.12.016

[27] C. Bergkvist, "Circular data analysis of repeated measurements: Inspired by growth hormone data," Ph.D. dissertation, Mathematical and Computing Sciences, University of Gothenburg, Sweden, October 2003.

[28] J. C. Bezdek, R. Ehrlich, and W. Full, "FCM: The fuzzy *c*-means clustering algorithm," *Computers and Geosciences*, vol. 10, no. 2, pp. 191–203, 1984. [Online]. Available: https://doi.org/10.1016/0098-3004(84)90020-7

[29] R. M. Andrews, I. Kubacka, P. F. Chinnery, R. N. Lightowlers, D. M. Turnbull, and N. Howell, "Reanalysis and revision of the Cambridge reference sequence for human mitochondrial DNA," *Nature Genetics*, vol. 23, no. 2, pp. 147–147, 1999. [Online]. Available: https://doi.org/10.1038/13779

[30] L. Katsir and O. Bahar, "Genome sequence of "Candidatus Carsonella ruddii" strain BT from the psyllid Bactericera trigonica," *Genome Announcements*, vol. 6, no. 4, pp. e01 466–17, 2018.

[31] BioProject accession number PRJNA544530 in the NCBI BioProject database. [Online]. Available: https://www.ncbi.nlm.nih.gov/bioproject/?term=PRJNA544530

[32] BioProject accession number PRJNA266911 in the NCBI BioProject database. [Online]. Available: https://www.ncbi.nlm.nih.gov/bioproject/?term=PRJNA266911

[33] Homo sapiens mitochondrion, complete genome - Nucleotide - NCBI. [Online]. Available: https://www.ncbi.nlm.nih.gov/nucleotide/NC_012920.1

[34] Candidatus Carsonella ruddii strain BT chromosome - Nucleotide - NCBI. [Online]. Available: https://www.ncbi.nlm.nih.gov/nuccore/NZ_CP024798.1

[35] ASM1346337v1 - Genome - Assembly - NCBI. [Online]. Available: https://www.ncbi.nlm.nih.gov/assembly/GCF_013463375.1

[36] Y. Wang, Y. Wang, C. Lang, D. Wei, P. Xu, and J. Xie, "Genome sequence of Lactobacillus curieae CCTCC M 2011381T, a novel producer of gamma-aminobutyric acid," *Microbiology Resource Announcements*, vol. 3, no. 3, pp. e00 552–15, 2015. [Online]. Available: https://doi.org/10.1128/genomeA.00552-15

[37] ASM78510v2 - Genome - Assembly - NCBI. [Online]. Available: https://www.ncbi.nlm.nih.gov/assembly/GCF_000785105.2

Tathagata Debnath received the Bachelor of Technology degree in computer science and engineering from the National Institute of Technology, Agartala, Tripura, India. He completed the Masters of Technology degree in computer science and engineering from Tripura University, a central university in India with the highest scores. He is pursuing a PhD degree from the Department of Computer Science at New Mexico State University (NMSU), USA. He has received a PhD tuition scholarship at NMSU. His research interests include genomics, proteomics, proteogenomics, bioinformatics, biological network analysis, computer vision, image processing, and machine learning including deep neural networks.

Mingzhou Song received the BS degree in electrical engineering from the Beijing University of Posts and Telecommunications, and the MS and PhD degrees from the Department of Electrical Engineering, University of Washington at Seattle. He was an assistant professor in the Department of Computer Science, Queens College of City University of New York. In 2005, he joined New Mexico State University, where he is a professor in the Department of Computer Science and a faculty member in the Graduate Program in Molecular Biology and Interdisciplinary Life Sciences. In 2019, he received a Fulbright scholar award and visited Charles University and Czech Technical University in Prague, Czech Republic. His research interests include statistical foundations for pattern discovery, data science algorithms for network inference, and applications to molecular biological systems. The two most popular software packages developed by his lab are 'Ckmeans.1d.dp' and 'FunChisq', both freely available to the public.