In-Database Time Series Clustering

YUNXIANG SU, Tsinghua University, China
KENNY YE LIANG, Tsinghua University, China
SHAOXU SONG*, BNRist, Tsinghua University , China

Time series data are often clustered repeatedly across various time ranges to mine frequent subsequence patterns from different periods, which could further support downstream applications. Existing state-of-the-art (SOTA) time series clustering method, such as K-Shape, can proficiently cluster time series data referring to their shapes. However, in-database time series clustering problem has been neglected, especially in IoT scenarios with large-volume data and high efficiency demands. Most time series databases employ LSM-Tree based storage to support intensive writings, yet causing underlying data points out-of-order in timestamps. Therefore, to apply existing out-of-database methods, all data points must be fully loaded into memory and chronologically sorted. Additionally, out-of-database methods must cluster from scratch each time, making them inefficient when handling queries across different time ranges. In this work, we propose an in-database adaptation of SOTA time series clustering method K-Shape. Moreover, to solve the problem that K-Shape cannot efficiently handle long time series, we propose Medoid-Shape, as well as its in-database adaptation for further acceleration. Extensive experiments are conducted to demonstrate the higher efficiency of our proposals, with comparable effectiveness. Remarkably, all proposals have already been implemented in an open-source commodity time series database, Apache IoTDB.

CCS Concepts: • **Information systems → Database query processing**; • **Computing methodologies → Machine learning**.

Additional Key Words and Phrases: time series clustering, database query processing

ACM Reference Format:
Yunxiang Su, Kenny Ye Liang, and Shaoxu Song. 2025. In-Database Time Series Clustering. *Proc. ACM Manag. Data* 3, 1 (SIGMOD), Article 46 (February 2025), 26 pages. https://doi.org/10.1145/3709696

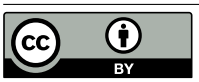
1 Introduction

Time series clustering is of great importance for analysis. For example, time series clustering could assist pattern mining of daily stock prices in finance [33], serve anomalous subsequence detection for yearly climate analysis in meteorology [21], facilitate the analysis of the characteristics associated with sleep apnea [25] and so on. The state-of-the-art (SOTA) time series clustering method K-Shape [31, 32] can proficiently cluster time series by shapes and achieve significantly better accuracy than other existing time series clustering methods.

However, K-Shape unfortunately faces challenges when meeting IoT scenarios, where extensive time series data stored in databases pose serious challenges for time series clustering. On the one hand, the arrival of IoT data is often out-of-order, due to transmission issues or sensor failures [15]. Most commodity time series databases employ Log Structured Merged Tree (LSM-Tree) [30] to

*Shaoxu Song (https://sxsong.github.io/) is the corresponding author.

Authors' Contact Information: Yunxiang Su, Tsinghua University, China, suyx21@mails.tsinghua.edu.cn; Kenny Ye Liang, Tsinghua University, China, liangy24@mails.tsinghua.edu.cn; Shaoxu Song, BNRist, Tsinghua University , China, sxsong@tsinghua.edu.cn.



ACM 2836-6573/2025/2-ART46
https://doi.org/10.1145/3709696

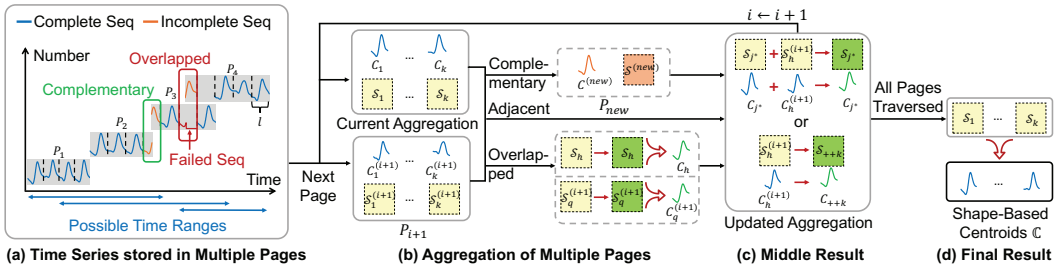


Fig. 1. The workflow of in-database K-Shape

handle such out-of-order arrivals, where data points are batched into files with possibly overlapped time intervals on disk. Therefore, applying the existing K-Shape algorithm typically involves loading all the data outside databases and sorting chronologically. Unfortunately, it is obviously inefficient in terms of I/O and incurs extra time.

On the other hand, time series clustering often needs to be performed multiple times over different time ranges in IoT databases to discover various patterns across different time periods. For instance, a forging machine in a steel processing factory may handle different types of steel bars during various time periods, depending on the production orders. When processing each type of steel bar, the machine may exhibit several working patterns influenced by its operating conditions and the environment. To identify all machine working patterns across different product types, analysts need to cluster subsequences of time series data over various time intervals to achieve a comprehensive clustering result. Apparently, loading all related data out of databases and clustering from scratch each time is extremely time-consuming.

1.1 Challenges

Considering such IoT scenarios with time series databases, the challenges of in-database time series clustering are as follows.

(1) Time series are often stored out-of-order in LSM-Tree databases owing to delayed arrivals, while existing clustering methods require data in chronological order. It incurs extra pre-processing time overheads to sort data chronologically. Therefore, we propose in-database adaptations for existing clustering method K-Shape, which can handle the out-of-order issues in LSM-Tree.

(2) Clustering may need to be performed repeatedly with different time filters for different tasks. K-Shape requires clustering from scratch each time, making it inefficient. To support frequent clustering queries, we propose in-database K-Shape with pre-computation.

(3) K-Shape is also inefficient in handling long subsequences due its high complexity with respect to the subsequence length. Therefore, we propose Medoid-Shape and in-database Medoid-Shape for acceleration, by leveraging approximate clustering and avoiding time-consuming eigenvector decomposition.

To tackle these challenges, the paper mainly focuses on *how to efficiently cluster time series in databases by leveraging database properties.* We primarily concentrate on achieving high efficiency for in-database time series clustering. The following example illustrates the challenges of LSM-Tree based database storage.

Example 1. *Figure 1(a) presents a time series stored in an LSM-Tree based time series database, recording the solar radiation intensity in a wind farm, where the x-axis denotes time and the y-axis denotes page number. Data points are batched into 4 pages P_1, P_2, P_3, P_4 on disk, each denoted by a gray rectangle. The higher the page number is, the later the data points arrive in the database. The dotted*

lines in Figure 1(a) split the data points in each page into several subsequences, and each subsequence represents the data of one day, i.e., starting from 00:00 and ending at 23:59. Most subsequences contain complete daily data (in blue), i.e., complete subsequences, while some only contain a part of daily data (in orange), i.e., incomplete subsequences (which will be formally defined in Section 2.3).

Generally, the order of data arriving in the database is the same as that of data generation, e.g., P_1, P_2 and P_3 have consecutive timestamps. Unfortunately, due to transmission issues, there may exist out-of-order points. For instance, in the failed subsequence of P_3 (in red), values are wrongly recorded as the default value 0 due to failure. The correct values are received after P_3 has been written to disk, and thus they are batched with the later page with a higher number 4.

Such out-of-order issues obviously impede in-database clustering. First, to apply out-of-database methods, all related pages on disk must be costly loaded and merged to ensure each data point is up-to-date. Moreover, users may cluster multiple times on different time ranges (denoted by blue arrows in Figure 1(a)). Existing methods can only repeatedly load data, and cluster from scratch each time.

Note that simply merging the clustering centroids of each page may lead to wrong results. For instance, the aforementioned failed subsequence may be taken into account when clustering in P_3, and the failed subsequence may form a new cluster by itself.

1.2 Contributions

To tackle challenges in IoT scenarios, in this work, we focus on the in-database adaptation of a SOTA time series clustering method, K-Shape [31, 32]. Our contributions are as follows:

(1) In-database K-Shape in Section 3. Unlike K-Shape [31] which clusters from scratch, in-database K-Shape utilizes pre-computed page-level metadata in Figure 1(a) for acceleration. Metadata, recording centroids and matrices, are later aggregated from each page for clustering. Due to disorder storage in databases, page-level metadata cannot be directly merged. Lemma 1, Propositions 2 and 3 address different cases of aggregating page metadata, as illustrated in Figure 1(b). We iteratively aggregate the next page and obtain the middle aggregation result in Figure 1(c). The final result is reached once all pages are processed, as depicted in Figure 1(d).

(2) Medoid-Shape in Section 4. The eigenvector decomposition stage of K-Shape [31] faces difficulties with long subsequences. Instead, we replace the time-consuming decomposition with an approximate medoid-based solution, while still achieving effective shape-based clustering results. The approximation correctness is theoretically ensured by Proposition 6, with the error bound guaranteed by Proposition 7.

(3) In-database Medoid-Shape in Section 5. Similar to the in-database K-Shape, the proposed Medoid-Shape can also be adapted to in-database Medoid-Shape for further acceleration. Again, the page metadata cannot be directly merged due to disorder. Propositions 8, 9 and 10 address various aggregation cases in LSM-Tree based storage accordingly.

(4) Extensive experiments in Section 6. We conduct extensive experiments to demonstrate the high efficiency of our proposals. In-database K-Shape shows up to 2 orders of magnitude improvement over the original K-Shape. Medoid-Shape and its in-database adaptation also demonstrate significantly greater efficiency than K-Shape, particularly as subsequence length increases.

Table 1 lists the time and space complexity of the proposals and the baselines. Here $iter$ denotes the maximal iteration numbers, W denotes the approximate clustering time complexity, s denotes the sampling size, r denotes the approximate cluster number, ℓ denotes the average overlapped length, and N, M denote the total and overlapped page numbers, respectively. Table 2 lists other symbols frequently used.

Table 1. Time and space complexity of methods

Method	Query Time	Space
K-Shape	$O(\max\{nkl\log l, nl^2, kl^3\} \times iter)$	$O(nl + kl^2)$
Medoid-Shape	$O(W + k^2srl\log l)$	$O(sl + kl)$
in-db K-Shape	$O(k(N-M)l^2 + kMl^3\lceil \ell/l \rceil)$	$O(kNl^2)$
in-db Medoid-Shape	$O(r(N-M)l + rM\ell + k^2srl\log l)$	$O(kNl)$

Table 2. Notations

Symbol	Description
n	the number of subsequences
l	subsequence length
k	cluster number
τ	time interval
X_i	a subsequence starting from $il\tau$ ending at $(i+1)l\tau$
$\mathbb{T}_l$	set of subsequences X_i with length l in time series T
$\mathbb{C}$	set of shape-based centroids
$C_j^{(i)}$	the j-th shape-based centroid in page P_i
$\mathcal{S}_j^{(i)}$	the j-th sum matrix in page P_i
$\delta_j^{(i)}$	the j-th average intra-cluster distance in page P_i
$U_j^{(i)}$	the j-th approximate centroid in page P_i
$w_j^{(i)}$	weight w.r.t. $U_j^{(i)}$

2 Preliminary

2.1 K-Shape Clustering

Shape-based distance [31, 32] measures the similarity by aligning two time series using cross-correlation, defined as follow.

Definition 1 (Shape-Based Distance [31, 32]). *For two time series X, Y, their Shape-Based Distance (SBD) is*

$$\mathrm{SBD}(X,Y) = 1 - \max_w \frac{(X * Y)_w}{\|X\|\|Y\|},$$

where operator $*$ denotes cross-correlation, a.k.a. vector convolution.

K-Shape clustering problem [31, 32] aims to find a partition for subsequences to maximize the overall intra-cluster similarities, with respect to the shape-based distance, defined as follow.

Definition 2 (K-Shape Clustering [31, 32]). *Given a set of subsequences $\mathbb{T}_l$, K-Shape clustering problem aims to find a set of centroids $\mathbb{C} = \{C_1, \ldots, C_k\}$, satisfying*

$$\mathbb{C}^* = \arg\max_{\mathbb{C}} \sum_{X_i \in \mathbb{T}_l} \max_{C_j \in \mathbb{C}} \left(\max_w \frac{(X_i * C_j)_w}{\|X_i\|\|C_j\|} \right). \tag{1}$$

Formula 1 aims to find centroids $\mathbb{C}$ to minimize the sum shape-based distances between each subsequence X_i and its closest centroid C_j, i.e., minimizing $\sum_{X_i \in \mathbb{T}_l} \mathrm{SBD}(X_i, C_j)$. Since minimizing

the sum of $\mathrm{SBD}(X_i, C_j)$ is equal to maximizing the sum of $\max_w \frac{(X_i * C_j)_w}{\|X_i\|\|C_j\|}$ referring to Definition 1, the problem of K-Shape clustering can be thus formulated as Definition 2.

The existing solution for K-Shape clustering iteratively assigns each point to its nearest centroid and updates each centroid referring to its members, until convergence. Each time updating the centroid, it extracts shapes from a sum matrix (see [31, 32] for more details), which utilizes matrix eigenvector decomposition, taking up to $O(kl^3)$ time, where l denotes the subsequence length.

Complexity Analysis. The overall time complexity for K-Shape is $O(\max\{nkl \log l, nl^2, kl^3\} \times iter)$, where n denotes the subsequence numbers, $iter$ denotes the maximum number of iterations. Thus, the efficiency of K-Shape is quite concerning when l increases.

2.2 LSM-Tree Database

To handle intensive writings in IoT scenarios, time series databases often employ LSM-Tree [30] as storage structure, such as InfluxDB [4] and Apache IoTDB [1]. Inserted data points are first stored in memory. When the number of points exceeds a threshold, they are batched and flushed to disk as an immutable file, i.e., a page. Note that the points in each page are chronologically sorted, but there may be conflicts or overlaps in timestamps across pages. For example, in the overlapped part of Figure 1(a), the data points in P_4 overwrite those in P_3 with the same timestamps.

Considering the possible out-of-order points in time series databases, a straightforward approach to apply K-Shape is to load all related points into memory, sort them by their timestamps, and then cluster subsequences from scratch. However, this approach may be extremely time-consuming for large volumes of time series data.

2.3 In-Database Time Series Clustering

Different from the previous works, we focus on in-database clustering, by leveraging pre-computed metadata in databases. Given a time series T stored in a database, in-database time series clustering aims to cluster all the subsequences of T cropped by scrolling windows, formally defined as follows.

Definition 3 (Subsequence). *Given subsequence length l, a time series T can be divided into a number of subsequences $\mathbb{T}_l = \{X_i\}$,*

$$X_i = \{p \in T | il\tau \le p.t < (i+1)l\tau\},$$

where τ denotes the time interval and i is a non-negative integer.

If subsequence X_i has $|X_i| = l$, it is called a complete subsequence. Otherwise, it is called an incomplete subsequence. The subsequence length l is set based on the real-world applications. For example, clustering the radiation intensity data by 24 hours in Example 1.

3 In-Database K-Shape Clustering

3.1 Metadata in A Single Page

Referring to Definition 3, a single page may contain multiple complete subsequences with $|X_i| = l$, and up to 2 incomplete subsequences with $|X_i| < l$. For complete subsequences in a page, we apply the K-Shape algorithm in [31] to find k clusters as well as the centroids through iterations. Then, we store all the centroids C_j as metadata. Besides, for all members in each cluster, we store the sum of all product matrices of each subsequence and its transpose, i.e., $\mathcal{S}_j = \sum_{X_i \in j\text{-th cluster}} X_i^T X_i$, $j = 1, \ldots, k$ as metadata. We also calculate the average intra-cluster shape-based distance δ_j for each cluster and store as metadata. For up to 2 incomplete subsequences, their values would also be stored, and would be further involved in the aggregation of complementary pages in Section 3.2.2.

3.2 Aggregation in Multiple Pages

To aggregate the pre-computed metadata of multiple pages, we sort all pages referring to their starting time and merge one by one. As illustrated in Figure 1(b), there may be three cases when merging a new page into the current aggregation result: adjacent (page P_2), complementary (page P_3), and overlapped (page P_4). We will further explain and clarify the three cases for aggregation, in Section 3.2.1, 3.2.2 and 3.2.3, respectively.

3.2.1 Aggregation of Adjacent Pages. Page P_{i+1} is an adjacent page, if $P_i.\mathsf{endTime}+\tau = P_{i+1}.\mathsf{startTime} \wedge P_{i+1}.\mathsf{startTime}\%(l\tau) = 0$. For instance, page P_2 in Figure 1(a) is an adjacent page, since the timestamps of P_1 and P_2 are chronologically consecutive, and both the tail of P_1 and the head of P_2 are complete subsequences.

With the matching strategy proposed by [10], the clustering centroids of an adjacent page could be directly merged into the current aggregation result by updating the corresponding sum matrices. To be specific, for each centroid $C_h^{(i+1)}$ in P_{i+1}, if there exists a centroid C_j in current aggregation result close enough to $C_h^{(i+1)}$, we merge them and update the corresponding sum matrix.

Lemma 1. *Given current aggregation result aggregated from $P_1, \ldots, P_i$, with centroids C_j, average intra-cluster distances δ_j and sum matrices $\mathcal{S}_j$, $j = 1, \ldots, k$, if P_{i+1} is an adjacent page, it could be aggregated into current aggregation result by: for each centroid $C_h^{(i+1)}$ in P_{i+1},*

$$j^* = \arg\min_j \mathsf{SBD}(C_j, C_h^{(i+1)}),$$
$$\mathcal{S}_{j^*} \leftarrow \mathcal{S}_{j^*} + \mathcal{S}_h^{(i+1)}, \ \textit{if}\ \mathsf{SBD}(C_{j^*}, C_h^{(i+1)}) \le \delta_{j^*},$$
$$\mathcal{S}_{++k} \leftarrow \mathcal{S}_h^{(i+1)}, \ \textit{otherwise.}$$

Example 2. *Figure 2 illustrates an example aggregating an adjacent P_{i+1} into the current aggregation result. Note that each centroid corresponds to a sum matrix which is omitted for more space in the figure. We first take $C_2^{(i+1)}$ for example. First, C_3 in current aggregation result is the closet centroid to $C_2^{(i+1)}$, i.e., $3 = \arg\min_j \mathsf{SBD}(C_j, C_2^{(i+1)})$. Moreover, they are close enough, i.e., $\mathsf{SBD}(C_3, C_2^{(i+1)}) < \delta_3$. Thus, they can be directly merged. Unfortunately, there does not exist a C_j in the current aggregation result close enough to $C_3^{(i+1)}$, so $C_3^{(i+1)}$ becomes a new centroid C_4 in the updated aggregation result.*

Unlike the existing matching strategy [10], we approximately update C_j and δ_j by weighted averaging to avoid time-consuming shape extraction. As the sum matrices are precisely updated, we only need to extract the centroids from the sum matrices once to obtain the final clustering centroids, after all pages are aggregated, as illustrated in Figure 1(d). Note that aggregation by Lemma 1 may create a new cluster, for example C_4 in Figure 2. To ensure there are k ultimate centroids, we will merge close centroids after merging all pages.

3.2.2 Aggregation of Complementary Pages. Page P_{i+1} is a complementary page, if $P_i.\mathsf{endTime}+\tau = P_{i+1}.\mathsf{startTime} \wedge P_{i+1}.\mathsf{startTime}\ \%(l\tau) \neq 0$. For instance, page P_3 in Figure 1(a) is a complementary page, where the green rectangle denotes the complementary part with the former page P_2. Though the timestamps of P_2 and P_3 are chronologically consecutive, the tail of P_2 and the head of P_3 are both incomplete subsequences. Intuitively, the complementary case could be inducted to the adjacent case, by further considering the newly formed subsequence as a new page.

Proposition 2. *Given current aggregation result aggregated from $P_1, \ldots, P_i$, if P_{i+1} is a complementary page, it could be aggregated into current aggregation result by sequentially aggregating P_{new} and P_{i+1} into current aggregation result by Lemma 1, where P_{new} only contains the newly formed*

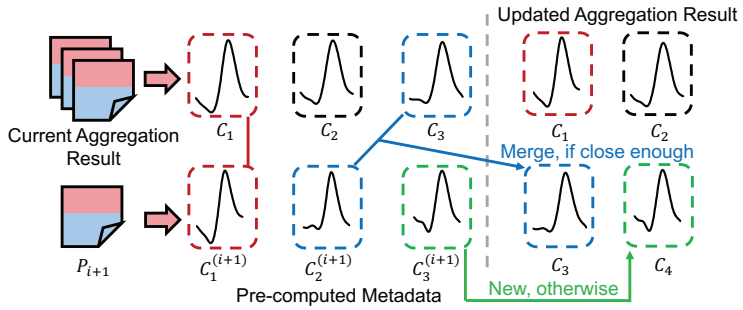


Fig. 2. The case for aggregating an adjacent page P_{i+1}

centroid $C^{(new)} = \{p \in P_i | p.t \geq \lfloor \frac{P_i.\text{endTime}}{l\tau} \rfloor l\tau\} \cup \{p \in P_{i+1} | p.t \leq \lceil \frac{P_{i+1}.\text{startTime}}{l\tau} \rceil l\tau\}$, *average distance* $\delta^{(new)} = 0$ *and sum matrix* $\mathcal{S}^{(new)} = {C^{(new)}}^T C^{(new)}$.

Example 3. *Figure 1(b) illustrates three cases for aggregation of multiple pages. To aggregate complementary page P_3 into the current aggregation result, we first combine the tail of P_2 and the head of P_3 to form a new page P_{new}, which contains only one centroid $C^{(new)}$ in orange (namely the new subsequence) and its sum matrix $\mathcal{S}^{(new)}$. Then we can obtain the aggregation result by sequentially merging two adjacent pages P_{new} and P_3 into the current aggregation result according to Lemma 1.*

3.2.3 Aggregation of Overlapped Pages . An overlapped page refers to a page having overlapped time intervals with its former page, i.e., $P_{i+1}.\text{startTime} \leq P_i.\text{endTime}$. In such case, some data points in P_i will be updated by the points with the same timestamps in P_{i+1}. For example, in Figure 1(a), the points of P_4 in the overlapped part update those of P_3, as P_4 has the higher number 4.

The current aggregation result w.r.t. $P_1, \ldots, P_i$ would be unreliable if there is an overlapped page P_{i+1}, and it should be updated by considering all conflicting points in P_i and P_{i+1}. That is, we need to load the points in the overlapped part of P_i and P_{i+1}, and then update the aggregation result. Fortunately, the aggregation result could be fast updated by the proposition below.

Proposition 3. *Given current aggregation result aggregated from $P_1, \ldots, P_i$, consider page P_{i+1}, overlapped with P_i. Denote X_j an updated subsequence belonging to the h-th cluster in current aggregation result. It is updated to X'_j belonging to the q-th cluster in P_{i+1}. The sum matrices can be updated by*

$$\mathcal{S}_h \leftarrow \mathcal{S}_h + \nabla X_j^T X_j + X_j^T \nabla X_j + \nabla X_j^T \nabla X_j,$$
$$\mathcal{S}_q^{(i+1)} \leftarrow \mathcal{S}_q^{(i+1)} - (X'_j)^T X'_j,$$

where $\nabla X_j = X'_j - X_j$, if X'_j a complete subsequence; otherwise, update $\mathcal{S}_h$ by treating X'_j as $C^{(new)}$ in Proposition 2.

Proof. Referring to Section 3.1, we have $\mathcal{S}_h = \sum_{X_i \in h\text{-th cluster}} X_i^T X_i$. If X_j is updated to X'_j, we need to eliminate the influence of X_j by subtracting $X_j^T X_j$ from $\mathcal{S}_h$. And then we need to consider the influence of X'_j by adding ${X'_j}^T X'_j$. Since the updates often happen on a small number of points, we can directly focus on the updates of X_j, and treat $X'_j = X_j + \nabla X_j$. Thus, we have new sum matrix with $\mathcal{S}_h \leftarrow \mathcal{S}_h - X_j^T X_j + {X'_j}^T X'_j = \mathcal{S}_h - X_j^T X_j + (X_j + \nabla X_j)^T (X_j + \nabla X_j) = \mathcal{S}_h + \nabla X_j^T X_j + X_j^T \nabla X_j + \nabla X_j^T \nabla X_j$. For sum matrix $S_q^{(i+1)}$, we only need to eliminate the influence of X'_j by subtracting ${X'_j}^T X'_j$.

Note that if X'_j is an incomplete subsequences, its sum matrices are not pre-computed as stated in Section 3.1. So in this case, Proposition 2 can be directly applied for this case to update and aggregate incomplete subsequences. □

To eliminate the negative influence of failed subsequences (e.g., the failed subsequence in Figure 1(a)), centroids and average intra-cluster distances should be re-calculated after updating sum matrices. After handling all conflicting points and updating all sum matrices by Proposition 3, overlapped cases could simply be inducted to adjacent cases in Lemma 1.

Example 4. *Consider page P_4 overlapped with P_3 in Figure 1(a). We first update the sum matrices S_h in current aggregation result, by loading original data points in P_3 and P_4 and applying Proposition 3 for each updated subsequence. Since there is no complete updated subsequence, we do not need to update the metadata in P_4. We then re-calculate centroid C_h and δ_h from $\mathcal{S}_h$ by extracting shapes. Then we could merge P_4 into the current aggregation result by Lemma 1.*

3.2.4 Discussion on Handling Shifting. First, in complementary cases, a new subsequence $C^{(\text{new})}$ is first assembled from two complementary pages as shown in Figure 1(a). Then, the shift between the newly generated subsequence $C^{(\text{new})}$ and the existing centroids $C_1, \ldots, C_k$ will be handled. After that, we will calculate shape-based distance (i.e., inner product calculations in [10]) and do the computation of new centroids, as shown in Figure 1(b).

Besides, shifting across pages may also happen across out-of-order cases, that is, overlapped cases. In this case, an existing subsequence X_j will be updated by a subsequence X'_j in a later page. During pre-computation, the shift of X_j has already been applied, referring to its corresponding pre-computed centroid C_h. Note that the updating from X_j to X'_j will not significantly change the shape of subsequences, so we apply the same shift to X'_j, update the sum matrix and then do the computation of the new centroid. Such a strategy is similar to the updating processing in [10], where shifts are not changed during centroid merging and updating.

3.3 In-Database K-Shape Algorithm

We are now prepared to introduce our in-database K-Shape. Algorithm 1 provides the pseudo-code, and Figure 1 illustrates the corresponding in-database adaptation workflow.

3.3.1 Algorithm. We first initialize the aggregation result with the first page P_1 in Line 2. Then we load the next page P_{i+1} each time, and merge with current aggregation result. For adjacent page P_{i+1}, we could directly merge P_{i+1} into current aggregation result by Lemma 1 in Line 6. If P_{i+1} is a complementary page, we first generate P_{new} with corresponding $C^{(new)}, \mathcal{S}^{(new)}, \delta^{(new)}$ by Proposition 2, and sequentially merge P_{new} and P_{i+1} into current aggregation result in Lines 8-9. For adjacent and complementary cases, we adopt weighted averaging to update centroids and intra-cluster distances in Line 10. Otherwise, if P_{i+1} is an overlapped page, we apply Proposition 3 on each updated subsequence, aggregate sum matrices, and then extract centroids by existing methods (namely ExtractShape) in [31, 32] from matrices from Lines 12 to 15. After traversing all pages, we extract final exact centroids in Line 16. If k has been updated as discussed in Section 3.2.1, we merge close centroids in Line 17 to ensure k ultimate centroids.

3.3.2 Complexity Analysis. Aggregating an adjacent or complementary page takes $O(kl^2)$ time. When aggregating an overlapped page, it takes $O(kl^3)$ time at most due to the time-consuming shape extraction. Denoting overall page number as N, overlapped page number as M and average overlapped length as ℓ, the overall time complexity for in-database K-Shape is $O(k(N-M)l^2 + kMl^3\lceil\frac{\ell}{l}\rceil)$.

Algorithm 1 K-Shape on LSM-Tree Based Store

Input: Multiple pages P_i each with pre-computed sum matrix $\mathcal{S}_i$
Output: Cluster centroids $\mathbb{C}$

```
sort all pages P_i by P_i.startTime
C_j ← C_j^(1), S_j ← S_j^(1), j = 1, ..., k
for each consecutive pages P_i, P_{i+1} do
    if P_i.endTime < P_{i+1}.startTime then
        if P_i.endTime + τ = P_{i+1}.startTime and P_{i+1}.startTime%(lτ) = 0 then
            update S_j with P_{i+1} by Lemma 1, j = 1, ..., k
        else
            generate P_new by Proposition 2
            update S_j with P_new, P_{i+1} by Lemma 1, j = 1, ..., k
        update C_j, δ_j by weighted averaging, j = 1, ..., k
    else
        for each updated subsequence do
            update corresponding S_h and S_q^(i+1) by Proposition 3
        update S_j with P_{i+1} by Lemma 1, j = 1, ..., k
        C_j, δ_j ← ExtractShape(S_j), j = 1, ..., k
C_j ← ExtractShape(S_j), j = 1, ..., k
merge close centroids if k updated
return ℂ ← {C_1, ..., C_k}
```

4 Medoid-Shape Clustering

Although K-Shape can outperform other time series clustering methods [31], both K-Shape and its in-database adaptation in Section 3 still meet difficulties with long subsequences. Therefore, we propose a variant of K-Shape clustering, Medoid-Shape clustering.

4.1 Medoid-Shape Clustering Problem

Why Medoid-Shape? When subsequence length l is large, in-database K-Shape takes $O(kl^2)$ space for each page to store sum matrices, and $O(kl^3)$ time to extract shapes from matrices. Thus, K-Shape and its in-database adaptation may take extremely high storage space and time costs with l increasing. Unlike K-Shape, Medoid-Shape clustering aims to find a subset of all subsequences to represent clustering centroids.

Definition 4 (Medoid-Shape Clustering). *Given a set of subsequences $\mathbb{T}_l$, Medoid-Shape clustering problem aims to find a set of centroids $\mathbb{C} = \{C_1, \ldots, C_k\} \subseteq \mathbb{T}_l$, satisfying*

$$\mathbb{C}^* = \arg\max_{\mathbb{C} \subseteq \mathbb{T}_l} \sum_{X_i \in \mathbb{T}_l} \max_{C \in \mathbb{C}} \left(\max_w \frac{(X_i * C)_w}{\|X_i\| \|C\|} \right).$$

Similar to Definition 2, Medoid-Shape clustering also aims to minimize the sum shape-based distances between each subsequence X_i and its closest centroid C_j. However, instead of finding an optimal arbitrary set of centroids $\mathbb{C}$, Medoid-Shape aims to choose a set of actual subsequences as centroids, i.e., $\mathbb{C} \subseteq \mathbb{T}_l$. Our solution below can significantly benefit from such simple modification.

4.2 A Greedy Solution

In this section, we first prove that the aim function of Medoid-Shape clustering is submodular in Proposition 4. Then an effective greedy solution in [27] could be applied with a promising error bound.

Given a subsequence X and a centroid $C \in \mathbb{C}$, we introduce $g(X, C) = \max_w \frac{(X*C)_w}{\|X\|\|C\|}$, equal to $1 - \mathrm{SBD}(X, C)$ referring to Definition 1, and we denote $f_{\mathbb{T}_l}(\mathbb{C}) = \sum_{X \in \mathbb{T}_l} \max_{C \in \mathbb{C}} g(X, C)$ as the aim function in Definition 4. For simplicity, we omit the subscript and use $f(\mathbb{C})$ to represent $f_{\mathbb{T}_l}(\mathbb{C})$.

Proposition 4. *Function $f(\mathbb{C})$ is a non-negative submodular function with respect to $\mathbb{C}$.*

Proof. Consider $\mathbb{A} \subseteq \mathbb{B}$ and a new element $V \notin \mathbb{B}$.

$$\begin{aligned} f(\mathbb{A} \cup \{V\}) - f(\mathbb{A}) &= \sum_X \max_{C \in \mathbb{A} \cup \{V\}} g(X, C) - \max_{C \in \mathbb{A}} g(X, C) \\ &= \sum_X \max\left(g(X, V), \max_{C \in \mathbb{A}} g(X, C)\right) - \max_{C \in \mathbb{A}} g(X, C). \end{aligned}$$

Similarly, for set $\mathbb{B}$,

$$f(\mathbb{B} \cup \{V\}) - f(\mathbb{B}) = \sum_X \max\left(g(X, C), \max_{C \in \mathbb{B}} g(X, C)\right) - \max_{C \in \mathbb{B}} g(X, C).$$

For an arbitrary X, denote $diff(X) = \max\left(g(X, V), \max_{C \in \mathbb{A}} g(X, C)\right) - \max_{C \in \mathbb{A}} g(X, C) - \max\left(g(X, V), \max_{C \in \mathbb{B}} g(X, C)\right) + \max_{C \in \mathbb{B}} g(X, C)$.

Since $\mathbb{A} \subseteq \mathbb{B}$, there has $\max_{C \in \mathbb{B}} g(X, C) \geq \max_{C \in \mathbb{A}} g(X, C)$. Consider the following three cases:

Case (1): $g(X, V) > \max_{C \in \mathbb{B}} g(X, C)$. There has

$$\max\left(g(X, V), \max_{C \in \mathbb{A}} g(X, C)\right) = \max\left(g(X, V), \max_{C \in \mathbb{B}} g(X, C)\right) = g(X, V).$$

Thus, $diff(X) = \max_{C \in \mathbb{B}} g(X, C) - \max_{C \in \mathbb{A}} g(X, C) \geq 0$.

Case (2): $\max_{C \in \mathbb{A}} g(X, C) \leq g(X, V) \leq \max_{C \in \mathbb{B}} g(X, C)$. Similarly, there has $diff(X) = g(X, V) - \max_{C \in \mathbb{A}} g(X, C) \geq 0$.

Case (3): $g(X, V) < \max_{C \in \mathbb{A}} g(X, C)$. There has $diff(X) = 0$.

By considering the three cases above, there always has $f(\mathbb{A} \cup \{V\}) - f(\mathbb{A}) \geq f(\mathbb{B} \cup \{V\}) - f(\mathbb{B}), \forall \mathbb{A} \subset \mathbb{B}, V \notin \mathbb{B}$. Thus, function f is submodular. □

Since f is submodular, we could apply existing greedy-stochastic method [27], i.e., pseudo-code in the following Algorithm 2, to find an approximately optimal set $\mathbb{C}$ to maximize $f(\mathbb{C})$.

Algorithm 2 greedily finds one centroid each iteration in Line 3. During one iteration, a set of subsequences $\mathbb{T}_l^s$ with size s is first sampled in Line 4. Then each subsequence in $\mathbb{T}_l^s$ is temporarily added to the current centroid set $\mathbb{C}$ in Line 5, and calculate the overall aim function f in Line 6. Then the subsequence in current sample set which can maximize the aim function is selected and added to centroids in Lines 7-8. By Algorithm 2, with subsequence number n and sample size s, we can find centroids in $O(k^2 snl \log l)$ time with $(1 - e^{-1} - e^{-\frac{sk}{n}})$ approximation ratio in expectation (see [27] for more details).

Note that shifting becomes part of the greedy solution. Because when calculating shape-based distances between subsequences and current centroids (as in Line 6 in Algorithm 2), shifting has already be taken into consideration.

Algorithm 2 Greedy Solution for Medoid-Shape Clustering

Input: A time series T, subsequence length l, cluster size k
Output: A set of cluster centroids $\mathbb{C}$

1: $\mathbb{T}_l \leftarrow$ all the subsequences in T with length l
2: $\mathbb{C} \leftarrow \phi$
3: **while** $|\mathbb{C}| < k$ **do**
4: sample $\mathbb{T}_l^s \subset \mathbb{T}_l \backslash \mathbb{C}$
5: **for** each subsequence $X \in \mathbb{T}_l^s$ **do**
6: $f(\mathbb{C} \cup \{X\}) \leftarrow \sum_{X_j \in \mathbb{T}_l} \max_{C_h \in \mathbb{C} \cup \{X\}} g(X_j, C_h)$
7: $X^* \leftarrow \arg\max_{X \in \mathbb{T}_l^s} f(\mathbb{C} \cup \{X\})$
8: $\mathbb{C} \leftarrow \mathbb{C} \cup \{X^*\}$
9: **return** $\mathbb{C}$

4.3 Approximate Aim Function Evaluation

Note that we need to traverse all pairs of subsequences X_j and current centroids C_h to evaluate the aim function $f(\mathbb{C} \cup \{X\})$ in Line 6 of Algorithm 2, with time complexity up to $O(kn)$ each time, which is extremely time-consuming. Thereby, we propose to find an approximate solution for aim function evaluation.

4.3.1 Intuition for Approximation. Consider two segments $T[1:n]$ and $T[2:n+1]$ in time series T. They tend to have similar shapes, i.e., close in shape-based distance in Definition 1, since there is only one element shifted. Further consider an arbitrary sequence C_h. Intuitively, the shape-based distance between C_h and $T[1:n]$ would be similar to that between C_h and $T[2:n+1]$. Therefore, it is natural to wonder whether there exists a triangle inequality for shape-based distances.

Unfortunately, regular triangle inequality cannot hold with only shape-based distance involved. A simple counter-example is $a = [1, 1], b = [-1, -1], c = [1, -1]$. There has $\max(a * b) = -1, \max(a * c) = 1$ and $\max(b * c) = 1$. Referring to Definition 1, $\mathrm{SBD}(a, b) = 1.5$, $\mathrm{SBD}(a, c) = 0.5$ and $\mathrm{SBD}(b, c) = 0.5$. Obviously, $\mathrm{SBD}(a, c) + \mathrm{SBD}(b, c) < \mathrm{SBD}(a, b)$, so the triangle inequality cannot hold. However, when both L_1-norm and L_2-norm involve, we can propose a bound for shape-based distances. Before that, we briefly introduce Lemma 5 as the mathematical basis of the bounds in Proposition 6.

Lemma 5 (Young's Inequality[41]). *Given $p, q, r \in [1, \infty]$, if $a \in L^p(\mathbb{R}^l), b \in L^r(\mathbb{R}^l)$, then*

$$\|a * b\|_q \leq \|a\|_p \|b\|_r,$$

where p, q, r satisfy $\frac{1}{q} + 1 = \frac{1}{p} + \frac{1}{r}$, L denotes the Lebesgue space and operator $*$ denotes cross-correlation, a.k.a. vector convolution.

Proposition 6. *Given two subsequences X_i and X_j, and a reference subsequence C_h, the difference between the shape-based distances of X_i and X_j can be bounded by*

$$g(X_j, C_h) - g(X_i, C_h) \leq \left(\frac{\|\nabla X\|}{\|X_i\| - \|\nabla X\|} \right) \left(g(X_i, C_h) + \frac{\|C_h\|_1}{\|C_h\|} \right),$$

where $\nabla X = X_j - X_i$ and $\|\bullet\|$ without subscript denotes the L_2-norm for simplicity.

Proof. Denote ∇X as the difference of subsequences X_i and X_j,

$$\nabla X = X_j - X_i. \tag{2}$$

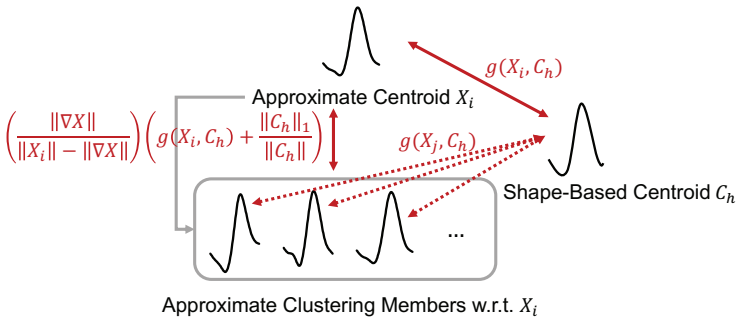


Fig. 3. The bound between $g(X_i, C_h)$ and $g(X_j, C_h)$

Since the convolution operation satisfies the linear property, the following equation holds,

$$X_j * C_h = X_i * C_h + \nabla X * C_h.$$

Therefore,

$$\max_w (X_j * C_h)_w = \max_w (X_i * C_h + \nabla X * C_h)_w \leq \max_w (X_i * C_h)_w + \max_w (\nabla X * C_h)_w. \tag{3}$$

Referring to the definition of infinity norm, we have

$$\max_w (\nabla X * C_h)_w \leq \|\nabla X * C_h\|_\infty.$$

Referring to the Young's inequality in Lemma 5, with $p = 1, r = \infty, q = \infty$, the following inequality holds,

$$\max_w (\nabla X * C_h)_w \leq \|\nabla X * C_h\|_\infty \leq \|\nabla X\|_\infty \|C_h\|_1. \tag{4}$$

That is, given a reference subsequence C_h, the term $\max_w (\nabla X * C_h)_w$ can be bounded by $\|\nabla X\|_\infty$. On the other hand, from Equation 2, there has

$$\begin{aligned} \|X_i\| &\leq \|X_j\| + \|\nabla X\|, \\ \frac{1}{\|X_j\|} &\leq \frac{1}{\|X_i\| - \|\nabla X\|}. \end{aligned} \tag{5}$$

By combining Equations 3, 4 and 5, there has

$$\frac{\max_w (X_j * C_h)_w}{\|X_j\|} \leq \frac{\max_w (X_i * C_h)_w}{\|X_i\| - \|\nabla X\|} + \frac{\|\nabla X\|_\infty \|C_h\|_1}{\|X_i\| - \|\nabla X\|},$$

that is,

$$\begin{aligned} \frac{\max_w (X_j * C_h)_w}{\|X_j\|\|C_h\|} - \frac{\max_w (X_i * C_h)_w}{\|X_i\|\|C_h\|} &\leq \frac{\|\nabla X\|}{\|X_i\| - \|\nabla X\|} \frac{\max_w (X_i * C_h)_w}{\|X_i\|\|C_h\|} + \frac{\|\nabla X\|_\infty}{\|X_i\| - \|\nabla X\|} \frac{\|C_h\|_1}{\|C_h\|} \\ &\leq \frac{\|\nabla X\|}{\|X_i\| - \|\nabla X\|} \left(\frac{\max_w (X_i * C_h)_w}{\|X_i\|\|C_h\|} + \frac{\|C_h\|_1}{\|C_h\|} \right) \\ &= \text{const.} \left(\frac{\|\nabla X\|}{\|X_i\| - \|\nabla X\|} \right). \end{aligned}$$

Note that the constant implies that it does not relate to X_j, it is only related to $g(X_i, C_h)$, the L_1-norm and L_2-norm of C_h. □

4.3.2 Implication of Proposition 6. Figure 3 illustrates the relationship among X_i, X_j and C_h. Generally, Proposition 6 holds for any X_i, X_j with small $\|\nabla X\|$ and any C_h. To better clarify the implication of Proposition 6, we consider X_i as a L_2-norm clustering centroid (namely approximate centroid), X_j as an arbitrary member w.r.t. X_i, and C_h as a shape-based centroid in current $\mathbb{C}$. The implication of Proposition 6 is that given approximate centroid X_i, the shape-based distances between its any member X_j and C_h can be bounded by $g(X_i, C_h) + \left(\frac{\|\nabla X\|}{\|X_i\| - \|\nabla X\|}\right)\left(g(X_i, C_h) + \frac{\|C_h\|_1}{\|C_h\|}\right)$. Without Proposition 6, we need to costly traverse all pairs of X_j and C_h (denoted by the dotted arrows in Figure 3) to evaluate aim function $f(\mathbb{C} \cup \{X\})$ in Line 6 of Algorithm 2. Instead, with Proposition 6, we could approximately evaluate $f(\mathbb{C} \cup \{X\})$ by just traversing all pairs of C_h and approximate centroids X_i. Since the number of approximate centroids is much lower than that of subsequences, the aim function evaluation time cost could significantly decrease.

4.3.3 Approximate Algorithm. Algorithm 3 presents the pseudo-code for approximate aim function evaluation, which serves to replace Line 6 of Algorithm 2 for acceleration.

Algorithm 3 Approximate Aim Function Evaluation

Input: An aim function f w.r.t. subsequences $\mathbb{T}_l$, current cluster centroids $\mathbb{C}$, a newly added centroid X

Output: Approximation for $f(\mathbb{C} \cup \{X\})$

1: **if** $\mathbb{T}_l$ not approximately clustered **then**
2: cluster all the subsequences $\mathbb{T}_l$ into r approximate clusters with centroids $U_1, ..., U_r$ respectively
3: $w_i \leftarrow$ number of members in the i-th cluster, $i = 1, \ldots, r$
4: $f(\mathbb{C} \cup \{X\}) \leftarrow 0$
5: **for** each centroid U_i **do**
6: $f(\mathbb{C} \cup \{X\}) \leftarrow f(\mathbb{C} \cup \{X\}) + w_i \max_{C_h \in \mathbb{C} \cup \{X\}} g(U_i, C_h)$
7: **return** $f(\mathbb{C} \cup \{X\})$

Algorithm 3 first applies approximate clustering on all subsequences in Lines 1-3, to cluster subsequences $\mathbb{T}_l$ into approximate clusters. Note that such approximate clustering only needs to be done once, and the results could be reused. Then the aim function f w.r.t. the current centroid set $\mathbb{C} \cup \{X\}$ can be calculated by only considering all approximate centroids $U_1, \ldots, U_r$ in Line 6.

Note that numerous existing clustering methods could be applied for approximate clustering in Line 2, such as grouping by a given threshold [37], K-Means [24] and LSH-based clustering [11]. Moreover, approximate clustering only serves as an intermediate metric for aim function evaluation, and does not directly correlate to the final shape-based clustering results.

4.3.4 Complexity Analysis. Algorithm 3 takes $O(krl \log l)$ time per evaluation, much lower than $O(knl \log l)$ without approximation. The overall time complexity for Medoid-Shape is $O(W + k^2 srl \log l)$, where $O(W)$ denotes the approximate clustering time.

4.3.5 Guaranteed Error Bound. In addition to lower time complexity, there is a guaranteed error bound in expectation in Proposition 7, whereas K-Shape cannot provide a promising error bound.

Proposition 7. *Algorithm 2 with approximation in Algorithm 3 returns an approximate maximum aim function $\tilde{f}$ with the following promising error bound in expectation*

$$\tilde{f} \geq (1 - e^{-1} - e^{-\frac{sk}{n}}) f^* - \sum_i \frac{w_i d_i}{\|U_i\| - d_i} \max_{C_h} \left(g(U_i, C_h) + \frac{\|C_h\|_1}{\|C_h\|} \right),$$

where f^* denotes the theoretical optimal aim function score, d_i denotes the maximum distance between members and centroid U_i in the i-th approximate cluster.

Proof. Given the maximum distance d_i between members and centroid w.r.t. U_i, we have $\|\nabla X\| \leq d_i$ for all members in the i-th approximate cluster. Since $\frac{\|\nabla X\|}{\|X_i\|-\|\nabla X\|}$ is monotonically increasing w.r.t. $\|\nabla X\|$, for any X_j belonging to the i-th approximate cluster we have

$$\frac{\|\nabla X\|}{\|U_i\| - \|\nabla X\|} \leq \frac{d_i}{\|U_i\| - d_i}.$$

Therefore, given any shape-based centroid C_h, by Proposition 6, there has

$$g(U_i, C_h) \geq g(X_j, C_h) - \frac{d_i}{\|U_i\| - d_i}\left(g(U_i, C_h) + \frac{\|C_h\|_1}{\|C_h\|}\right).$$

Furthermore, by considering all members for U_i, we have

$$w_i g(U_i, C_h) \geq \sum_{X_j} g(X_j, C_h) - \frac{w_i d_i}{\|U_i\| - d_i}\left(g(U_i, C_h) + \frac{\|C_h\|_1}{\|C_h\|}\right). \tag{6}$$

Note that the aim function score $\tilde{f}$ returned by Algorithms 2 and 3 is

$$\tilde{f} = \sum_i w_i \max_{C_h} g(U_i, C_h). \tag{7}$$

Moreover, referring to the error analysis in [27] with respect to greedy-stochastic method, given sample size s and subsequence number n there has

$$\sum_i \sum_{X_j} \max_{C_h} g(X_j, C_h) \geq (1 - e^{-1} - e^{-\frac{sk}{n}}) f^*, \tag{8}$$

where f^* denotes the theoretical optimal aim function score.

By combining Equations 6, 7 and 8, we can prove

$$\begin{aligned}
\tilde{f} &= \sum_i w_i \max_{C_h} g(U_i, C_h) \\
&\geq \sum_i \sum_{X_j} \max_{C_h} g(X_j, C_h) - \sum_i \frac{w_i d_i}{\|U_i\| - d_i}\left(g(U_i, C_h) + \frac{\|C_h\|_1}{\|C_h\|}\right) \\
&\geq (1 - e^{-1} - e^{-\frac{sk}{n}}) f^* - \sum_i \frac{w_i d_i}{\|U_i\| - d_i}\left(g(U_i, C_h) + \frac{\|C_h\|_1}{\|C_h\|}\right).
\end{aligned}$$

□

5 In-Database Medoid-Shape

To further accelerate Medoid-Shape in Section 4, the approximate centroids could also be stored as pre-computed metadata during flushing, just like in-database K-Shape in Section 3.

5.1 Metadata in A Single Page

Different from Section 3.1, we do not store K-Shape clustering results as metadata. Instead, we apply approximate clustering on each page, such as aforesaid K-Means [24], LSH-based clustering [11] and so on. Approximate centroids U_j, cluster weights w_j and average intra-cluster distances δ_j are stored as metadata. Since there is no need to store matrices, the space complexity of in-database Medoid-Shape is much smaller than that of in-database K-Shape.

5.2 Aggregation in Multiple Pages

Similar to Section 3.2, we also consider three cases when merging a new page into current aggregation result: adjacent pages in Section 5.2.1, complementary pages in Section 5.2.2, and overlapped pages in Section 5.2.3.

5.2.1 Aggregation of Adjacent Pages. Similar to Section 3.2.1, to aggregate adjacent pages, we may merge approximate centroids close enough into one centroid, or new a centroid.

Proposition 8. *Given current aggregation result aggregated from $P_1, \ldots, P_i$, with approximate centroids U_j and cluster sizes w_j, $j = 1, \ldots, r$, if P_{i+1} is an adjacent page, it could be aggregated into current aggregation result by: for each centroid $U_h^{(i+1)}$ in P_{i+1},*

$$j^* = \arg\min_j \|U_j - U_h^{(i+1)}\|,$$

$$U_{j^*} \leftarrow \frac{w_{j^*} U_{j^*}}{w_{j^*} + w_h^{(i+1)}} + \frac{w_h^{(i+1)} U_h^{(i+1)}}{w_{j^*} + w_h^{(i+1)}}, \ \text{if } \|U_{j^*} - U_h^{(i+1)}\| \le \delta_{j^*},$$

$$U_{++r} \leftarrow U_h^{(i+1)}, \ \text{otherwise.}$$

After applying Proposition 8, cluster weights w_j and average distances δ_j could be accordingly updated by weighted averaging.

5.2.2 Aggregation of Complementary Pages. Similar to Section 3.2.2, we generate P_{new} by the proposition below, and sequentially merge P_{new} and P_{i+1} into current aggregation result.

Proposition 9. *Given current aggregation result aggregated from $P_1, \ldots, P_i$, if P_{i+1} is a complementary page, it could be aggregated into current aggregation result by sequentially aggregating P_{new} and P_{i+1} into current aggregation result by Proposition 8, where P_{new} only contains the newly formed centroid $U^{(new)} = \{p \in P_i | p.t \ge \lfloor \frac{P_i.\text{endTime}}{l\tau} \rfloor l\tau\} \cup \{p \in P_{i+1} | p.t \le \lceil \frac{P_{i+1}.\text{startTime}}{l\tau} \rceil l\tau\}$, $w^{(new)} = 1$ and average distance $\delta^{(new)} = 0$.*

5.2.3 Aggregation of Overlapped Pages. Similar to Section 3.2.3, we only need to update each subsequence involved in the overlapped part, and induct overlapped cases into adjacent cases.

Proposition 10. *Given current aggregation result aggregated from $P_1, \ldots, P_i$, consider page P_{i+1}, overlapped with P_i. Denote X_j an updated subsequence belonging to the h-th cluster in current aggregation result. It is updated to X'_j belonging to the q-th cluster in P_{i+1}. The approximate centroids can be updated by*

$$U_h \leftarrow U_h + \frac{1}{w_h}(X'_j - X_j), \quad U_q^{(i+1)} \leftarrow \frac{w_q}{w_q - 1} U_q^{(i+1)} - \frac{1}{w_q - 1} X'_j.$$

5.3 In-Database Medoid-Shape Algorithm

On the basis of metadata and aggregation introduced in Sections 5.1, 5.2, we are now ready to introduce in-database Medoid-Shape. Algorithm 4 provides the pseudo-code, and Figure 4 illustrates the corresponding in-database adaptation workflow.

5.3.1 Algorithm. We first initialize the aggregation result with the first page P_1 in Line 2. Then we iteratively load the next page P_{i+1} , and merge it with the current aggregation result. For an adjacent page P_{i+1}, we can directly merge P_{i+1} into the current aggregation result by Proposition 8 in Line 6. If P_{i+1} is a complementary page, we generate P_{new} by Proposition 9, and sequentially merge both P_{new} and P_{i+1} into the current aggregation result in Lines 8-9. In both adjacent and

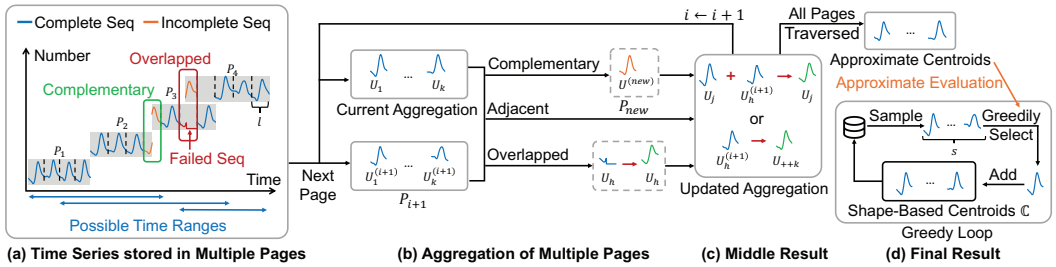


Fig. 4. The workflow of in-database Medoid-Shape

Algorithm 4 Medoid-Shape on LSM-Tree Based Store

Input: A time series T stored in multiple pages P_i, each with pre-computed approximate clustering centroids $U_j^{(i)}, j = 1, \ldots, r$

Output: Medoid-Shape cluster centroids $\mathbb{C}$

1: sort all pages P_i by P_i.startTime
2: $U_j \leftarrow U_j^{(1)}, j = 1, \ldots, r$
3: **for** each consecutive pages P_i, P_{i+1} **do**
4: **if** P_i.endTime $< P_{i+1}$.startTime **then**
5: **if** P_i.endTime $+ \tau = P_{i+1}$.startTime and P_{i+1}.startTime$\%(l\tau) = 0$ **then**
6: update U_j with P_{i+1} by Proposition 8, $j = 1, \ldots, r$
7: **else**
8: generate P_{new} by Proposition 9
9: update U_j with P_{new}, P_{i+1} by Proposition 8, $j = 1, \ldots, r$
10: update U_j, δ_j by weighted averaging, $j = 1, \ldots, r$
11: **else**
12: **for** each updated subsequence **do**
13: update corresponding U_h and $U_q^{(i+1)}$ by Proposition 10
14: update U_j with P_{i+1} by Proposition 8, $j = 1, \ldots, r$
15: solve $\mathbb{C}$ by Algorithms 2 with 3
16: **return** $\mathbb{C}$

complementary cases, U_j and δ_j can be updated by weighted averaging in Line 10. Otherwise, if P_{i+1} is an overlapped page, we apply Proposition 10 on each updated subsequence, and then merge it as in the adjacent case in Lines 12-14. After traversing all pages, we apply a greedy solution with approximation (Algorithms 2 and 3) to determine the final centroids in Line 15. Note that for in-database adaptation, the sample set can be approximated as as the union of all approximate centroids from all pages, to avoid the costly process of loading original data.

5.3.2 Complexity Analysis. Each time aggregating an adjacent or complementary page takes $O(rl)$ time, where r denotes the number of approximate clusters. Given average overlapped length ℓ, aggregating an overlapped page takes $O(r\ell)$ time. The final greedy selection with approximation takes $O(k^2srl \log l)$ time referring to the complexity analysis in Section 4.3.3. Consider N pages in total with M overlapped pages, the overall time complexity for in-database Medoid-Shape is $O(r(N-M)l + rM\ell + k^2srl \log l)$.

Table 3. Dataset Information

Dataset	Subsequence Length	# Subsequences	# Clusters
GW-WindTurbine	192	50,000	3
TY-Vehicle	96	100,000	3
UCR-Air	166	4,307	3
UCR-ECG	140	5,000	5

6 Experiments

6.1 Setup

We compare the original K-Shape [31, 32] as the baseline with proposed Medoid-Shape and our in-database adaptations: in-database K-Shape, in-database Medoid-Shape. We implement all these methods in Apache IoTDB [1], an open-source LSM-Tree based time series database. For a fair comparison, K-Shape and Medoid-Shape are implemented as user defined functions by utilizing the series reader in the database, which merge all possibly overlapped pages, load all related data points and cluster from scratch.

Table 3 lists 4 datasets used in our evaluation, with 2 private datasets from our industrial partners and 2 public datasets available on UCR time-series collection [13]. We use the first 5,000 and 10,000 subsequences in 2 private datasets for all experiments except subsequence number experiments, due to the consuming time costs of K-Shape under large data sizes. Without extra instructions, the default page size in the database is set to 10240, the default max iteration number is set to 100, and the default sample rate is set to 0.3. For Medoid-Shape and its in-database adaptation, we use K-Means as the default approximate clustering method.

The rationale behind the choice of two benchmark datasets is that they are among the most complex (with a subsequence length of 166) and large-volume datasets (with 5,000 subsequences) in UCR repository. And the rationale behind the choice of private datasets is that the private datasets from our industry partners contain more subsequences (up to 100,000) with large length (up to 192), while public time series clustering datasets can hardly reach such data size. The original aim of evaluation on private datasets is to demonstrate the performance of our proposals under extremely large-volume data and long subsequences. It is worth noting that the evaluation can be also conducted on other public datasets from UCR repository with no doubt.

All the experiments run on a machine with Intel Core i7-10750H CPU (2.60GHz). The algorithm code has been included (by system developers) in the system repository of Apache IoTDB [2]. The experiment related code is available in [3].

6.2 Scalability under different data loads

6.2.1 Scalability in Subsequence Numbers. Figure 5 presents the time costs under different subsequence numbers n. Our in-database methods perform the best under all data sizes. The out-of-database methods apparently cost more time, since they need to load all data points and cluster from scratch. In-database K-Shape takes less time than in-database Medoid-Shape in Figures 5(b), 5(d). This is because the subsequence length of TY-Vehicle is much smaller than that of other datasets. And UCR-ECG has a higher cluster number than other datasets, leading to more loops for Medoid-Shape and in-database Medoid-Shape, as analyzed in Sections 4.3.3 and 5.2.3.

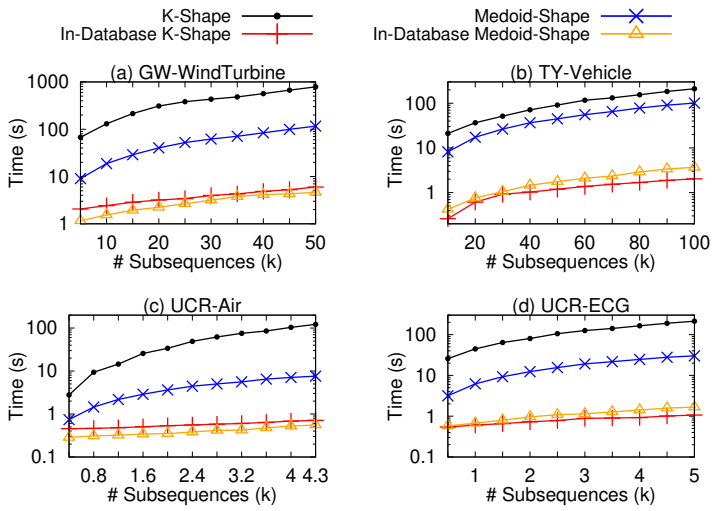


Fig. 5. Time costs under different subsequence numbers

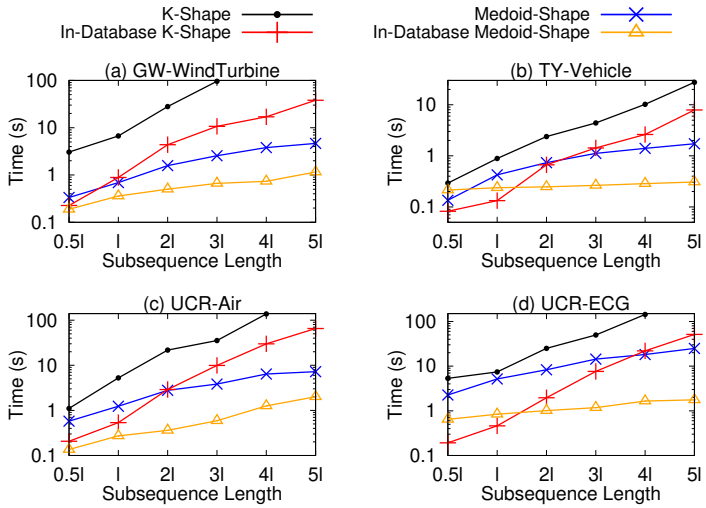


Fig. 6. Time costs under different subsequence lengths

6.2.2 Scalability in Subsequence Lengths. Figure 6 presents the time costs under different subsequence lengths, where l in the x-axis denotes the default subsequence length in Table 3. The subsequence numbers of 4 datasets are fixed to 400 in this experiment for a fair comparison, and we change subsequence length from $0.5l$ to $5l$. With the increase of subsequence lengths, both the time costs of K-Shape and in-database K-Shape increase rapidly, due to the time-consuming shape extraction (referring to analysis in Sections 2.1 and 3.3). However, Medoid-Shape and in-database Medoid-Shape respectively show up to 1 and 2 orders of magnitude improvement under large subsequence length, i.e., $5l$ with around 1000 data points in Figure 6(a)(c)(d). They are demonstrated to cost significantly less time given long subsequences.

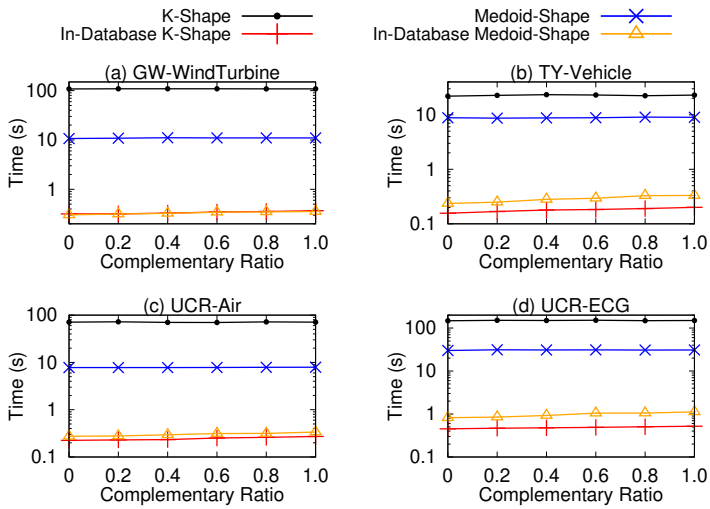


Fig. 7. Time costs under different complementary ratios

6.3 Efficiency under Different Configuration

6.3.1 Varying Complementary Page Numbers. Figure 7 presents the time costs under different complementary page numbers. We set page size as $100l$ instead of 10240 to control complementary page numbers. The x-axis in Figure 7 represents the ratio of complementary page number to overall page number. All pages are adjacent when the ratio is equal to 0, and all pages are complementary when the ratio is equal to 1. Our in-database proposals have a slight increase with the number of complementary pages increasing. This is because we need to create a new page containing one subsequence, which is generated by two complementary pages, as discussed in Sections 3.2.2. This newly formed page incurs an extra adjacent page aggregation, resulting in extra execution time. Still, the out-of-database methods have almost constant time costs.

6.3.2 Varying Overlapped Page Number. Figure 8 presents the time costs under different overlapped page numbers M. Similarly, the overlapped ratio represents the ratio of overlapped page number M to overall page number N. As shown, the time cost of in-database K-Shape has a rapid increase as the overlapped ratio increases, while that of in-database Medoid-Shape has a slight increase. This is because in-database K-Shape needs to extract shapes each time processing an overlapped page, which takes $O(kMl^3\lceil\frac{\ell}{l}\rceil)$ time, as discussed in Section 3.3. In contrast, in-database Medoid-Shape can efficiently process overlapped pages in linear time $O(rM\ell)$, as discussed in Section 5.3. Note that for "extreme" out-of-order cases where overlap ratio is equal to 1.0, i.e., every page is overlapped with some pages, in-database K-Shape significantly costs more time, but still less than K-Shape.

6.3.3 Varying Overlap Length. Figure 9 illustrates the time costs under different overlap lengths ℓ with a fixed overlapped ratio equal to 1.0. Overlap length ℓ denotes the average number of points with conflicts between two consecutive pages, i.e., points with the same timestamps but in different pages. The time costs of in-database adaptations have a slight increase when overlap length increases, since they need more time to split the overlapped part given more points involved in overlap, as analyzed in Sections 3.3 and 4.3. The last points in Figure 9 indeed represent the "extreme" out-of-order cases where overlapped ratio is equal to 1.0 and overlapped length reaches

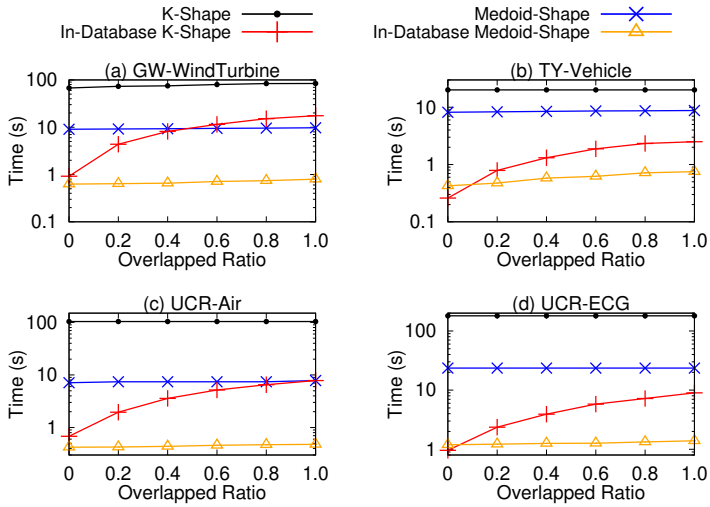


Fig. 8. Time costs under different overlapped ratios

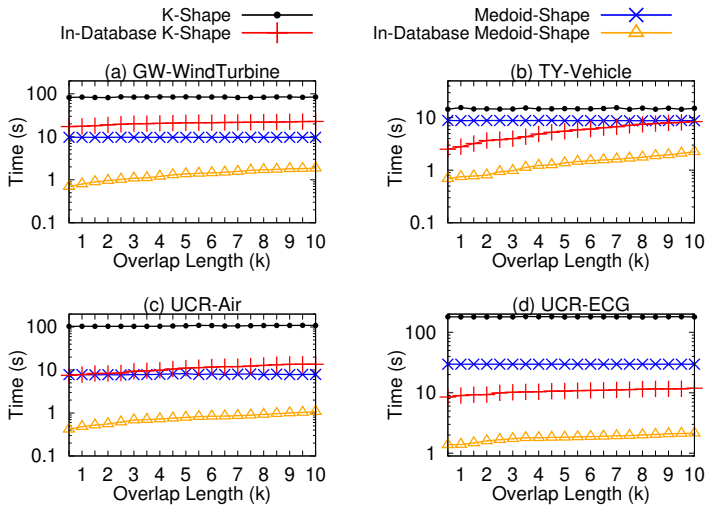


Fig. 9. Time costs under different overlap lengths

10,000 data points (w.r.t. 10,240 data points in a page). As shown, in such "extreme" cases, the in-database methods still outperform the according out-of-database methods in terms of efficiency.

6.4 Evaluation on Other Platforms

6.4.1 Evaluation on RocksDB. Indeed, our proposal is not dependent or relevant to a particular database management, as long as time series are segmented and stored in pages with pre-computation. Note that segmented storage is rather common in databases, and almost all commodity databases apply segmented store. Thus, our in-database solutions can also be applied to other DBMS with segmented storage, such as RocksDB.

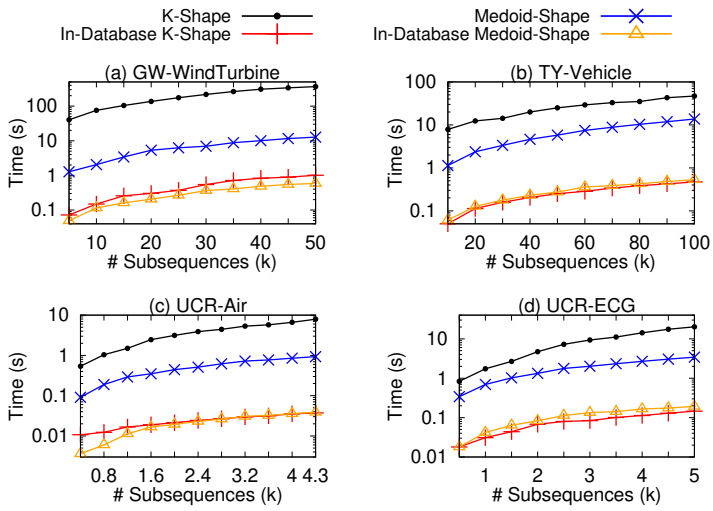


Fig. 10. Evaluation on RocksDB

We implement and compare the efficiency of our proposals on RocksDB [6]. RocksDB natively supports meta blocks, which record metadata and are stored in the same files with data points. Therefore, we utilize such meta blocks to store our pre-computed metadata in Sections 3.1 and 5.1.

Figure 10 presents the query time costs with varying subsequence numbers in RocksDB. Still, in-database methods significantly take less time than out-of-database methods. It demonstrates that our solution does not depend on or be relevant to a particular DBMS. Note that all the baselines become more efficient than the implementation on IoTDB. This is because RocksDB does not use a server-client architecture, but instead is invoked as header files. Besides, the efficient compilation in C language also contributes to the high efficiency.

Therefore, our solution can be easily deployed to LSM-Tree based DBMS with pre-computation, such as Apache IoTDB [1], while some DBMS may not natively support pre-computation, posing a deployment challenge. It demonstrates that the proposed techniques are not particularly relevant or dependent on the previously implemented system Apache IoTDB.

6.4.2 Evaluation on PostgreSQL. We consider LSM-Tree databases, since they are more difficult and expensive to conduct query processing and clustering analysis. To efficiently handle intensive writing of data, LSM-Tree database takes a strategy of out-of-place updates, which means that the updates of tuples are stored into another file with higher version. And a new tuple does not explicitly overwrite the obsolete tuple on disk. Instead, updates in traditional databases are often in-place. That is, when updating a tuple, traditional databases locate this tuple on disk (by timestamp) and overwrite its obsolete value. Therefore, the case in traditional databases, such as PostgreSQL [5], is relatively easier.

In this sense, it is easier to deploy our proposal to relational databases, as long as relational databases support pre-computation. To clarify and extend the scope of the proposed solutions, we further discuss the alternative about applying the proposed solution on relational databases. To better clarify this, we implement and evaluate our proposal in a relational database, PostgreSQL [5]. The implementation on PostgreSQL is a special case where all pages are adjacent and they can be aggregated referring to Section 3.2.1. Since PostgreSQL natively stores the statistical metadata

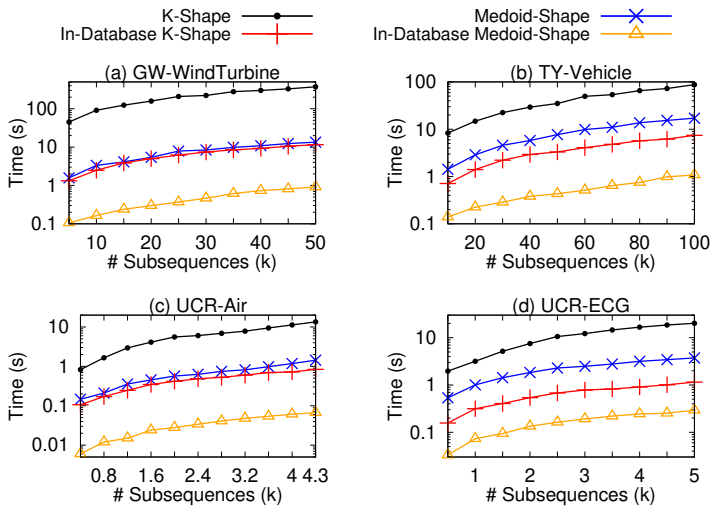


Fig. 11. Evaluation on PostgreSQL

as a relational table, we similarly store the pre-computed metadata of each block as a relational table. When a query arrives, our implementation loads all related tables and then aggregate.

Figure 11 shows the query time costs with varying subsequence numbers on PostgreSQL. In-database K-Shape on PostgreSQL costs more time than that on RocksDB, because it takes more time to query a relational table on PostgreSQL than loading metadata on RocksDB. Our in-database solutions consistently cost less time. The results extend the scope of the proposed solution and prove that our solution is not only crafted for LSM-Tree databases.

6.5 Evaluation on More Datasets and Baselines

6.5.1 Evaluation on Additional Baselines. We further conduct efficiency and effectiveness experiments with more recent advanced time series clustering methods. We implement and compare FrOKShape [23], MUSLA [42], wFCM [17], PAM+DTW [18, 35] and Time2Feat [9] as follows.

In Figures 12 and 13, we conduct critical differential diagrams to show the average ranking across datasets of each method w.r.t. time costs and Rand Indexes, respectively. Rand Index [34] is defined by $R = \frac{TP+TN}{TP+TN+FP+FN}$, where TP, TN, FP and FN denote true positive, true negative, false positive and false negative classification series, respectively. Each method in the figures associates with an average ranking. Higher ranking approaching to 1 means higher Rand Indexes or lower time costs. The wiggly line connects all measures that do not perform statistically differently according to the Nemenyi test [29].

It is worth noting that in-database time series clustering often needs to be repeatedly conducted on extensive data in LSM-Tree based store, which motivates our proposal as in Section 1. Therefore, our main focus is the efficiency of in-database clustering. As for effectiveness, in Figure 13, K-Shape and in-database K-Shape, as well as Medoid-Shape and in-database Medoid-Shape do not perform statistically differently in terms of Rand Indexes. Besides, our proposals can indeed achieve similar effectiveness as these time-consuming baselines.

As illustrated in Figure 12, all these additional methods cost more time than our proposal referring to the following reasons. FrOKShape [23] defines a multi-variable shape-based distance and utilizes DTW barycenter averaging as clustering strategy, which has similar time costs to

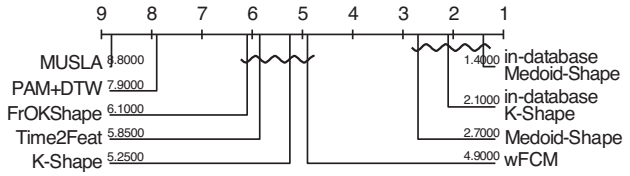


Fig. 12. Ranking of methods based on the average time costs across UCR datasets

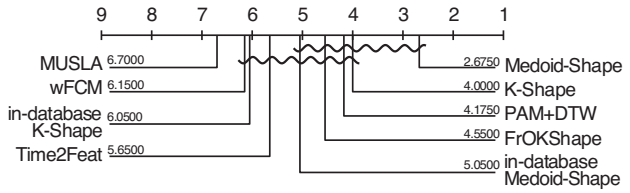


Fig. 13. Ranking of methods based on the average Rand Indexes across UCR datasets

K-Shape. MUSLA [42] learns shapelet-transformed representation, and the learning process is relatively time-consuming. wFCM [17] divides a sequence into multiple segments, applies linear regression on each segment and then uses weighted DTW [35] on the regression coefficients, which inevitably incurs extra time. Similarly, the high complexity of calculating DTW also impedes the efficiency of PAM+DTW [18, 35]. Time2Feat [9], as a learning method, extracts multiple features from subsequences, and clusters in the feature space. Besides, all these methods do not utilize database storage structure for acceleration. Besides, all these additional methods do not utilize database storage structure for acceleration, and therefore, they cost more time.

6.5.2 Evaluation on Additional Datasets. Moreover, to tackle the limited numbers of datasets and better clarify the significance of the results, we further consider 20 UCR datasets covering all the dataset types on UCR time-series collection [13], including Sensor, Simulated, Spectrum, Image, Device, Traffic and so on. We select 2 datasets from each type with varying characteristics (e.g., subsequence numbers, subsequence length and cluster numbers). Due to the limited space, the full experimental results are given in the supplementary [7].

These experimental results in Figure 12 show that, in-database K-Shape, Medoid-Shape and in-database Medoid-Shape consistently cost much less time across all datasets. Moreover, in Figure 13, K-Shape and in-database K-Shape, as well as Medoid-Shape and in-database Medoid-Shape achieve comparable average Rand Indexes, without statistical difference. These experiments on more public datasets from UCR further demonstrate the performance of our greedy approximation.

7 Related Works

Our proposals put effort in in-database time series clustering, while existing works scarcely focus on in-database clustering. To further highlight the novelty of our proposals, we review existing time series clustering methods in Section 7.1, which are devised without considering in-database scenarios, and recent in-database learning methods in Section 7.2.

7.1 Time Series Clustering

Time series clustering strongly relies on series similarity measures [8]. Three most commonly used measures are euclidean distance (ED) [14], dynamic time wrapping (DTW) [35] and shape-based

distance (SBD) [31, 32]. ED can be measured in linear time, but cannot handle shifts and aligning. DTW and its variants [43] can well capture shapes and shifts, while they require quadratic time per measure. SBD is proved to perform better than ED, and costs much less time than DTW. Besides, KL distance [12], related distance [22], HMM dependency [36] can also serve as measures, while either inefficient or ineffective.

Therefore, we focus on time series clustering with SBD in this work. Since SBD is defined with convolution, classic clustering prototypes cannot be directly applied to calculate centroids, such as K-Means [24], DBSCAN [16]. Thus, K-Shape [31, 32] proposes to calculate centroids by matrix eigenvector decomposition. Unfortunately, it may be extremely time-costly given long subsequences due to its cubic time complexity. Times-C [39] proposes to optimize K-Shape clustering by GPU, by utilizing CUDA programming. However, CUDA programming is dedicated to GPU resources, while difficult to adapt to database systems often without GPU resources. Although there exist clustering methods based on deep neural network [9, 19, 40], they all need abundant training time to perform, much more inefficient than K-Shape. Our proposals in this work aim to accelerate K-Shape in databases, as well as handle long subsequences.

7.2 In-Database Learning

In-database learning emerges as a prevailing trend in both database and machine learning areas in recent years. On the one hand, numerous learned data structures are proposed for databases for storage optimization and query acceleration. For example, learned index [26] proposes to replace traditional database indexes by a model learning data distribution. Learned bloom filter [28] utilizes a learned function to enhance the query performance.

On the other hand, machine learning models could benefit from databases for acceleration. For instance, existing work [20] accelerates statistical methods by utilizing synergizing key tools for relational databases. LSMAR [38] learns auto-regressive models in time series databases. However, to the best of our knowledge, there is no existing work studying in-database time series clustering.

8 Conclusion

In this paper, we study in-database time series clustering, to support repeatedly time series clustering with various time ranges in LSM-Tree based time series databases. Existing out-of-database methods unfortunately suffer from low efficiency, given IoT data of large volume and frequent clustering queries with different time filters. Therefore, we propose to efficiently cluster time series in databases, with the utilization of database properties. Specifically, we devise an in-database adaptation of a SOTA time series clustering method, K-Shape. To solve that K-Shape performs inefficiently with long subsequences, we propose Medoid-Shape and the according in-database Medoid-Shape adaptation for further acceleration. We derive several propositions to ensure the aggregation in multiple pages for in-database proposals. We also prove a guaranteed error bound for Medoid-Shape to ensure its effectiveness. Remarkably, we implement and deploy all proposals in Apache IoTDB, an open-source commodity LSM-Tree based time series database. Extensive experiments demonstrate the higher efficiency of our proposals with comparable effectiveness.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China (62232005, 62021002, 92267203, 62072265), the National Key Research and Development Plan (2021YFB3300500), Beijing National Research Center for Information Science and Technology (BNR2025RC01011), and Beijing Key Laboratory of Industrial Big Data System and Application. Shaoxu Song (https://sxsong.github.io/) is the corresponding author.

References

[1] 2024. Apache IoTDB. https://iotdb.apache.org.
[2] 2024. Code. https://github.com/apache/iotdb/tree/research/indb-ts-cluster/.
[3] 2024. Experimental code. https://github.com/indb-ts-cluster/indb-ts-cluster/.
[4] 2024. InfluxData. https://www.influxdata.com.
[5] 2024. PostgreSQL. https://www.postgresql.org.
[6] 2024. RocksDB. https://rocksdb.org.
[7] 2024. Supplementary. https://indb-ts-cluster.github.io/supplementary.pdf.
[8] Saeed Reza Aghabozorgi, Ali Seyed Shirkhorshidi, and Ying Wah Teh. 2015. Time-series clustering - A decade review. *Inf. Syst.* 53 (2015), 16–38.
[9] Angela Bonifati, Francesco Del Buono, Francesco Guerra, and Donato Tiano. 2022. Time2Feat: Learning Interpretable Representations for Multivariate Time Series Clustering. *Proc. VLDB Endow.* 16, 2 (2022), 193–201.
[10] Paul Boniol, John Paparrizos, Themis Palpanas, and Michael J. Franklin. 2021. SAND: Streaming Subsequence Anomaly Detection. *Proc. VLDB Endow.* 14, 10 (2021), 1717–1729.
[11] Moses Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *STOC*. ACM, 380–388.
[12] Rainer Dahlhaus. 1996. On the Kullback-Leibler information divergence of locally stationary processes. *Stochastic processes and their applications* 62, 1 (1996), 139–168.
[13] Hoang Anh Dau, Eamonn Keogh, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, Yanping, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, Gustavo Batista, and Hexagon-ML. 2018. The UCR Time Series Classification Archive. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.
[14] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. 1994. Fast Subsequence Matching in Time-Series Databases. In *SIGMOD Conference*. ACM Press, 419–429.
[15] Chenguang Fang, Shaoxu Song, and Yinan Mei. 2022. On Repairing Timestamps for Regular Interval Time Series. *Proc. VLDB Endow.* 15, 9 (2022), 1848–1860.
[16] Junhao Gan and Yufei Tao. 2015. DBSCAN Revisited: Mis-Claim, Un-Fixability, and Approximation. In *SIGMOD Conference*. ACM, 519–530.
[17] Hongyue Guo, Mengjun Wan, Lidong Wang, Xiaodong Liu, and Witold Pedrycz. 2024. Weighted Fuzzy Clustering for Time Series With Trend-Based Information Granulation. *IEEE Trans. Cybern.* 54, 2 (2024), 903–914.
[18] Christopher Holder, David Guijo-Rubio, and Anthony J. Bagnall. 2023. Clustering Time Series with k-Medoids Based Algorithms. In *AALTD@ECML/PKDD (Lecture Notes in Computer Science, Vol. 14343)*. Springer, 39–55.
[19] Dino Ienco and Roberto Interdonato. 2020. Deep Multivariate Time Series Embedding Clustering via Attentive-Gated Autoencoder. In *PAKDD (1) (Lecture Notes in Computer Science, Vol. 12084)*. Springer, 318–329.
[20] Mahmoud Abo Khamis, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. 2018. In-Database Learning with Sparse Tensors. In *PODS*. ACM, 325–340.
[21] Hardy Kremer, Stephan Günnemann, and Thomas Seidl. 2010. Detecting Climate Change in Multivariate Time Series Data by Novel Clustering and Cluster Tracing Techniques. In *ICDM Workshops*. IEEE Computer Society, 96–97.
[22] Joseph Lee Rodgers and W Alan Nicewander. 1988. Thirteen ways to look at the correlation coefficient. *The American Statistician* 42, 1 (1988), 59–66.
[23] Yucheng Li, Derong Shen, Tiezheng Nie, and Yue Kou. 2022. A new shape-based clustering algorithm for time series. *Inf. Sci.* 609 (2022), 411–428.
[24] James MacQueen et al. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1. Oakland, CA, USA, 281–297.
[25] Son T. Mai, Sihem Amer-Yahia, Sébastien Bailly, Jean Louis Pépin, Ahlame Douzal Chouakria, Ky T. Nguyen, and Anh-Duong Nguyen. 2018. Evolutionary Active Constrained Clustering for Obstructive Sleep Apnea Analysis. *Data Sci. Eng.* 3, 4 (2018), 359–378.
[26] Ryan Marcus, Emily Zhang, and Tim Kraska. 2020. CDFShop: Exploring and Optimizing Learned Index Structures. In *SIGMOD Conference*. ACM, 2789–2792.
[27] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrák, and Andreas Krause. 2015. Lazier Than Lazy Greedy. In *AAAI*. AAAI Press, 1812–1818.
[28] Michael Mitzenmacher. 2018. A Model for Learned Bloom Filters and Optimizing by Sandwiching. (2018), 462–471.
[29] Peter Bjorn Nemenyi. 1963. *Distribution-free multiple comparisons.* Princeton University.
[30] Patrick E. O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth J. O'Neil. 1996. The Log-Structured Merge-Tree (LSM-Tree). *Acta Informatica* 33, 4 (1996), 351–385.
[31] John Paparrizos and Luis Gravano. 2015. k-Shape: Efficient and Accurate Clustering of Time Series. In *SIGMOD Conference*. ACM, 1855–1870.

[32] John Paparrizos and Luis Gravano. 2017. Fast and Accurate Time-Series Clustering. *ACM Trans. Database Syst.* 42, 2 (2017), 8:1–8:49.
[33] Francesco Pattarin, Sandra Paterlini, and Tommaso Minerva. 2004. Clustering financial time series: an application to mutual funds style analysis. *Comput. Stat. Data Anal.* 47, 2 (2004), 353–372.
[34] William M Rand. 1971. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association* 66, 336 (1971), 846–850.
[35] Hiroaki Sakoe and Seibi Chiba. 1978. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing* 26, 1 (1978), 43–49.
[36] Padhraic Smyth. 1996. Clustering Sequences with Hidden Markov Models. In *NIPS*. MIT Press, 648–654.
[37] Shaoxu Song, Chunping Li, and Xiaoquan Zhang. 2015. Turn Waste into Wealth: On Simultaneous Clustering and Cleaning over Dirty Data. In *KDD*. ACM, 1115–1124.
[38] Yunxiang Su, Wenxuan Ma, and Shaoxu Song. 2023. Learning Autoregressive Model in LSM-Tree based Store. In *KDD*. ACM, 2061–2071.
[39] Xun Wang, Ruibao Song, Junmin Xiao, Tong Li, and Xueqi Li. 2023. Accelerating k-Shape Time Series Clustering Algorithm Using GPU. *IEEE Trans. Parallel Distributed Syst.* 34, 10 (2023), 2718–2734.
[40] Chenxiao Xu, Hao Huang, and Shinjae Yoo. 2021. A Deep Neural Network for Multivariate Time Series Clustering with Result Interpretation. In *IJCNN*. IEEE, 1–8.
[41] William Henry Young. 1912. On the multiplication of successions of Fourier constants. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* 87, 596 (1912), 331–339.
[42] Nan Zhang and Shiliang Sun. 2023. Multiview Unsupervised Shapelet Learning for Multivariate Time Series Clustering. *IEEE Trans. Pattern Anal. Mach. Intell.* 45, 4 (2023), 4981–4996.
[43] Jing Zhou, Shanfeng Zhu, Xiaodi Huang, and Yanchun Zhang. 2015. Enhancing Time Series Clustering by Incorporating Multiple Distance Measures with Semi-Supervised Learning. *J. Comput. Sci. Technol.* 30, 4 (2015), 859–873.

Received July 2024; revised September 2024; accepted November 2024