RTCM: A Distributed Snapshot-Based Framework for Real-Time Co-Movement Mining

Chenxu Wang*
Xi'an Jiaotong University
Xi'an, China
cxwang@mail.xjtu.edu.cn

Jiaxing Wei
Xi'an Jiaotong University
Xi'an, China
weijiaxing0803@stu.xjtu.edu.cn

Tianyi Li
Aalborg University
Aalborg, Denmark
tianyi@cs.aau.dk

Hongzhen Xiang
Xi'an Jiaotong University
Xi'an, China
hzxiang@stu.xjtu.edu.cn

Junzhou Zhao
Xi'an Jiaotong University
Xi'an, China
junzhou.zhao@mail.xjtu.edu.cn

Pinghui Wang
Xi'an Jiaotong University
Xi'an, China
phwang@mail.xjtu.edu.cn

Tao Qin
Xi'an Jiaotong University
Xi'an, China
qin.tao@mail.xjtu.edu.cn

Yushuai Li
Aalborg University
Aalborg, Denmark
yusli@cs.aau.dk

Christian S. Jensen
Aalborg University
Aalborg, Denmark
csj@cs.aau.dk

Abstract

Co-movement mining aims to discover a group of objects traveling together for a certain period of time. Real-time co-movement mining can facilitate important applications such as congestion control and movement prediction. However, most existing methods focus on mining co-movement patterns from offline trajectory datasets. Some distributed real-time methods adopt sliding window-based schemes for online co-movement mining, which often generate redundant overlapping sub-trajectories.

This paper presents RTCM, a distributed snapshot-based framework for real-time co-movement mining. By comparing clusters in the current snapshot with candidate co-movement patterns in the previous snapshot, RTCM avoids redundant sub-trajectory comparisons. First, we propose a snapshot-based method to mine co-movement patterns in parallel subtasks. Second, we propose a dynamic and adaptive partitioning strategy tailored for streaming scenarios to effectively mitigate partition skew. Meanwhile, we implement a distributed DBSCAN algorithm based on this partitioning strategy to enable efficient density-based clustering in streaming settings. Finally, we propose two pruning strategies that use a novel data structure to eliminate unnecessary cluster comparisons and employ a subsequence-based set intersection algorithm to accelerate cluster comparisons by skipping irrelevant subsequences. Experiments on four real-world datasets show that RTCM reduces computational latency and improves throughput by approximately 50% compared to three state-of-the-art baselines, while producing higher-quality co-movement patterns.

Keywords

Real-time Co-movement Mining, Distributed Object Clustering, Cluster Pruning

1 Introduction

Massive trajectory data are being generated that capture the movements of objects such as individuals and vehicles. It is increasingly important to extract meaningful patterns from the data efficiently, as this can provide valuable insights into the intrinsic dynamics of real-world movements. A pivotal pattern is the co-movement pattern [20, 43] that captures a group of objects moving together for a certain duration.

Real-time co-movement mining plays an important role in applications such as congestion control [7, 41, 42], movement prediction [15, 23, 25], traffic mode classification [14, 29, 32], and trajectory compression [13, 18, 19]. It benefits users by identifying public transportation passengers and potential carpool partners, and it helps urban traffic managers detect dense traffic and crowds for early intervention. Effective real-time co-movement mining requires low latency, as a snapshot must be processed before the arrival of the subsequent snapshot. To avoid data backlog caused by fluctuating data volumes (e.g., rush-hour trajectory surges), a quantitative latency threshold strictly less than 1/3 of the snapshot interval is established, ensuring sufficient buffer time for unexpected computational overhead and stable real-time performance. Most existing algorithms [9, 17, 27, 28, 37, 40] focus on offline use. Existing distributed online methods [5] rely on pattern enumeration to identify co-movement objects, which cannot strictly meet the requirements of latency threshold. However, three key challenges are still unresolved.

Challenge I: Identifying real-time co-movement patterns among numerous candidates.

A candidate co-movement represents a set of objects that move together in one or more snapshots. In practice, the number of clusters in a single snapshot can be large, yielding many candidate co-movements. In a single snapshot, if there are $|C|$ clusters and each cluster contains an average of $|O|$ objects, the number of candidate co-movement subsets that meet the minimum size m and are generated solely from the current snapshot can reach $\sum_{c \in C} \binom{|c|}{m}$, which exhibits super-linear growth.

Existing methods [5] utilize a sliding window to identify overlapping sub-trajectories between time windows. The sliding window partitions continuous, unbounded streaming data into a series of ordered, finite data segments, dividing streaming trajectories into sub-trajectories with explicit start and end timestamps. However, a large number of repeated sub-trajectories are often generated across time windows, and processing them in each individual window is redundant and time-consuming. For example, in Fig. 1, Window-1 and Window-2 overlap with sub-trajectories

10.48786/edbt.2026.38

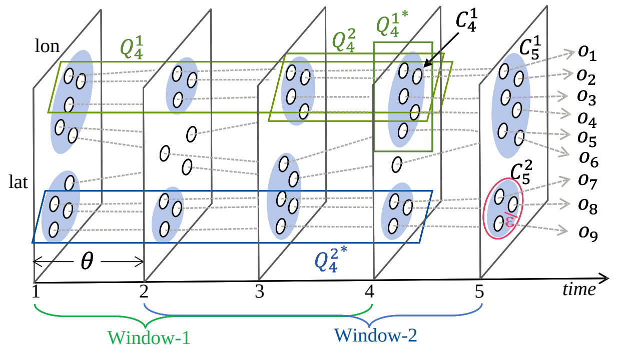


Figure 1: Streaming trajectories, clusters, and co-movement patterns.

of objects from time 2 to 4. Processing these overlapping sub-trajectories separately for each window results in unnecessary duplication and reduced efficiency.

Challenge II: Balanced partitioning for streaming data.

Partitioning refers to the process of dividing a large-scale trajectory dataset into smaller, non-overlapping partitions that can be processed independently by distributed worker nodes, with the goal of balancing computational loads and minimizing inter-node data transmission overhead.

The distribution of streaming trajectory data, such as traffic flow during rush hours, is non-stationary. Let the total number of objects be p and the number of worker nodes be N, with an ideal load per node of p/N. However, when partition skew occurs, the maximum partition load can reach $O(p)$, reducing the system throughput to the level of a single machine and thereby nullifying the scalability advantages of the distributed system. Therefore, it is essential to propose a dynamic and adaptive partitioning strategy suitable for streaming scenarios.

The core approach adopted in [24, 31] for addressing dynamic partition skew involves increasing the number of partitions for hot keys to balance the data volume across partitions, which introduces an additional shuffle operation. Although this method can effectively mitigate skew, it increases the overall number of partitions and adds additional overhead for data transmission between partitions.

Challenge III: Efficient cluster comparison between adjacent snapshots.

Given that the number of candidate co-movements is $|V|$ and the number of clusters is $|C|$, the naive enumeration method requires $|V| * |C|$ set comparison operations. As the cluster size increases, the computational complexity of this approach becomes prohibitively high, posing significant challenges to real-time processing efficiency.

Existing methods [5] enumerate all subsets in a cluster with m or more objects to identify objects present in consecutive time intervals. As the cluster size increases, the number of subsets grows rapidly, reducing the efficiency of co-movement generation.

We propose RTCM, a distributed $\underline{\text{r}}$eal-$\underline{\text{t}}$ime $\underline{\text{c}}$o-movement $\underline{\text{m}}$ining framework. To address *Challenge I*, we present a snapshot-based method for efficient co-movement mining in parallel subtasks, where subtask is determined by time and spatial dimensions. Unlike sliding-window-based methods, RTCM compares clusters in the current snapshot with candidate co-movement patterns from the previous snapshots in each subtask, thus avoiding overlapping sub-trajectory comparisons.

To address *Challenge II*, we propose a novel dynamic adaptive global partitioning strategy for streaming data, comprising two phases: initialization and adaptive adjustment. In initialization, objects are evenly distributed across partitions via latitude-based sorting. During adaptive adjustment, local and global counters monitor object volumes per partition and system-wide. If a partition exceeds a dynamically calculated threshold (based on the average volume per partition), boundary recalibration is triggered for the next snapshot. This approach maintains balance by adjusting partition boundaries without increasing their number, thus avoiding extra shuffling.

To address *Challenge III*, we design a novel data structure for candidate co-movement patterns to accelerate cluster intersection. Each candidate co-movement pattern consists of a primary candidate co-movement pattern and a list of sub-candidate co-movement patterns. Using this structure, we develop two pruning strategies to eliminate unnecessary cluster comparisons, by leveraging the observation that co-movement patterns do not change markedly in a short time duration. We also adopt a subsequence-based set-intersection method that divides large sets into smaller subsequences and only intersects those with common elements. This avoids redundant computations of cluster intersections and significantly improves efficiency.

Distributed computing is adopted primarily for its outstanding scalability, enabling seamless expansion with additional nodes to handle massive streaming trajectories and dynamic tidal data. This ensures consistent real-time performance even as data scale grows. For instance, in practical urban mobility applications, the registered ride-hailing vehicles in Beijing exceed 100,000 units, and a distributed framework can efficiently process the high-frequency trajectory data generated by these vehicles immediately, without latency spikes during peak hours.

The contributions of this paper are summarized as follows.

- We propose RTCM, a distributed real-time co-movement mining framework. It is the first snapshot-based method that leverages cluster comparisons between consecutive snapshots for efficient real-time co-movement patterns identification in parallel subtasks.
- We propose a dynamic and adaptive partitioning strategy that maintains the load balance for streaming data through two phases. It monitors partition loads and automatically rebalances the partitions when skew is detected, ensuring continuous processing while efficiently handling data fluctuations.
- We propose two pruning strategies to reduce unnecessary cluster intersections, enhancing the efficiency of candidate co-movement pattern generation.
- We conduct extensive experiments on four real-world datasets, finding that RTCM can improve latency and throughput by approximately 50% compared to three state-of-the-art baselines.

The rest of the paper is organized as follows. Section 2 defines the problem of real-time co-movement mining and presents preliminaries. Section 3 details the proposed method RTCM, and Section 4 reports the experimental results. Section 5 covers related work, followed by Section 6 concluding the work and offering future research directions.

2 Problem Definition

Let o be an object (e.g., a vehicle or a person) that can produce a continuous and unbounded streaming trajectory. Each object o is assigned a unique numeric ID, ID_o, at its first appearance.

Definition 1. ***(Streaming Trajectory)*** *A streaming trajectory $T_o = \langle p_0, p_1, \ldots, p_t, \ldots \rangle$ of object o is an unbounded time-ordered sequence, where $p_t = (l, t)$ represents the position of object o at time t, and $l = (lon,\ lat)$ is a tuple of longitude and latitude.*

Definition 2. ***(Snapshot)*** *A snapshot at time t is a set $S_t = \{o_i | 1 \le i \le n\}$, representing objects observed between t and $t + \theta$. Further, $o_i.l$ denotes the position of object o_i.*

Following existing studies[9, 17, 19, 37, 40], we assume that object positions are collected at a regular interval of θ, where θ is determined by specific applications. To avoid duplicate objects in a snapshot, we retain only the position closest to t for each object. θ can be set based on the sampling rate to minimize such issues.

We adopt DBSCAN to find object clusters for each snapshot. It has two critical parameters to assess the density of objects in an area: ε is the maximum distance of two neighboring objects, and *minPts* is the minimum number of objects in a qualified cluster. We adopt the classic DBSCAN algorithm [8] for object clustering, which has been established as the standard method for density-based co-movement pattern detection in subsequent research [5, 37].

Definition 3. ***(Core Point)*** *An object o_i is a core point if at least minPts points o_j satisfy $d(o_i, o_j) \le \varepsilon$, where $d(o_i, o_j)$ is the distance between o_i and o_j, and o_j can be o_i itself.*

Definition 4. ***(Density Reachable Point)*** *An object o_m is density reachable from o_1 if a chain of objects $\langle o_1, o_2, \ldots, o_m \rangle$ $(m \ge 2)$ exists such that i) $d(o_i, o_{i+1}) \le \varepsilon$ $(1 \le i < m - 1)$; and ii) o_i is a core point $(1 \le i < m)$.*

Definition 5. ***(Snapshot Cluster)*** *A snapshot cluster is a subset $C_t^i \subseteq S_t$ grouped by DBSCAN for snapshot S_t, where i is the index of a cluster. We denote the set of snapshot clusters for S_t by $\mathbb{C}_t = \{C_t^1, C_t^2, \ldots, C_t^n\}$.*

A co-movement pattern describes a group of objects that move together with certain spatio-temporal similarity, a concept foundational to trajectory data mining [17, 43]. Our work builds upon the formal definitions of co-movements established in prior research [5, 9]. Specifically, we extend these concepts by introducing a novel data structure for candidate co-movements to efficiently track their evolution across snapshots.

Definition 6. ***(Co-movement)*** *Given an integer m, an integer lifetime k, a consecutive constraint L and a connection constraint G, a co-movement pattern is defined as a tuple $Q = (O, T)$, where $O = \{o_1, o_2, \ldots, o_n\}$ is a set of accompanying objects and $T = (t_s, t_i, ..., t_e)$ is a time sequence. The object set O is clustered with each timestamp in the time series T from t_s to t_e. The time series T consists of multiple disjoint time segments, each comprising a continuous sequence of timestamps.*

Let $T_i (1 \le i \le m)$ be the time segments of time sequence T. (i) For each $|T_i| \ge L$, time sequence T satisfies $L - consecutive$. (ii) For each $T_{i+1}[start] - T_i[end] \le G$, time sequence T satisfies $G - connected$, where T_i and T_{i+1} are adjacent time segments, $T_{i+1}[start]$ is the starting snapshot of T_{i+1} and $T_i[end]$ is the ending snapshot of T_i.

Q satisfies: i) $t_e - t_s \ge k$, where $t_e - t_s$ is called the lifespan of Q; ii) $|O| \ge m$, $|O|$ is the number of objects. iii) The time sequence T satisfies $L - consecutive$ and $G - connected$.

Definition 7. ***(Candidate Co-movement)*** *A candidate co-movement $V_t = \langle Q_t{}^*, Q_t^1, Q_t^2, \ldots, Q_t^h \rangle$ is a sequence of co-movement patterns without the constraints of lifespan, where $Q_t{}^*$ is a primary candidate co-movement pattern, $Q_t{}^*.t_e = t$, and $Q_t{}^*.O$ is a snapshot cluster in the snapshot S_t. Q_t^i is a sub-candidate co-movement pattern of $Q_t{}^*$ due to the dynamic arrival and departure of objects: $Q_t^i.O \subset Q_t{}^*.O$, $Q_t^i.t_e = Q_t{}^*.t_e = t$, and $Q_t^i.t_s < Q_t{}^*.t_s$, i.e., a sub-candidate co-movement pattern starts earlier but has no more than accompanying objects than the primary candidate co-movement pattern.*

Let $\mathbb{V}_t = \{V_t^1, V_t^2, \ldots, V_t^n\}$ denote the set of candidate co-movements for snapshot S_t. Let $SubQ_t = \langle Q_t^1, Q_t^2, \ldots, Q_t^h \rangle$ denote the set of sub-candidate co-movements of $Q_t{}^*$. We sort $SubQ_t$ by starting time in ascending order, i.e., $Q_t^i.t_s \le Q_t^{i+1}.t_s$, to facilitate cluster pruning.

Definition 8. ***(Co-movement Mining)*** *Given the current snapshot S_t, the candidate co-movements $\{\mathbb{V}_{t-1}, \mathbb{V}_{t-2}, \ldots, \mathbb{V}_{t-G}\}$, an integer m, an integer lifetime k, a consecutive constraint L and a connection constraint G, co-movement mining is to derive the set of valid co-movements $\mathbb{Q}_t$ at snapshot S_t among all candidate co-movements. Real-time co-movement mining requires immediate output of all co-movements at each snapshot.*

Example 1. *Fig. 1 illustrates snapshot clusters of streaming trajectories. At time = 5, given minPts = 3, o_2, o_3, o_4, o_5 and o_8 are core points, while o_1, o_6, o_7 and o_9 are density reachable points. $C_5^1 = \{o_1, o_2, \ldots, o_6\}$ and $C_5^2 = \{o_7, o_8, o_9\}$ are clusters obtained by DBSCAN. Thus, $\mathbb{C}_5 = \{C_5^1, C_5^2\}$.*

Example 2. *In Fig. 1, $Q_4^{1*} = (\{o_1, o_2, o_3, o_4, o_5\}, [4, 4])$ and $Q_4^{2*} = (\{o_7, o_8, o_9\}, [1, 4])$ are primary candidate co-movements in S_4. $Q_4^1 = (\{o_1, o_2, o_3\}, [1, 4])$ and $Q_4^2 = (\{o_1, o_2, o_3, o_4\}, [3, 4])$ are two sub-candidate co-movements of Q_4^{1*}. $V_4^1 = \langle Q_4^{1*}, Q_4^1, Q_4^2 \rangle$ and $V_4^2 = \langle Q_4^{2*} \rangle$ are two candidate co-movements in snapshot S_4.*

Real-time co-movement mining first generates snapshot clusters for each snapshot by DBSCAN, and then generates all candidate co-movements for each snapshot by intersecting all snapshot clusters with the candidate co-movements from the previous snapshot. In the end, it mines real co-movements with a lifespan no less than k from these candidate co-movements for each snapshot. Compared to offline co-movement mining, we find co-movements at each snapshot for real-time mining.

Table 1: Frequently Used Notations.

Notations	Descriptions
o	An object
S_t	A snapshot at time t
ε	The distance threshold in DBSCAN
minPts	The minimal number of objects in DBSCAN
C_t^i	A cluster in snapshot S_t
$\mathbb{C}_t$	The set of clusters in snapshot S_t
m	The cardinality threshold
k	The consecutive snapshot threshold
L	The consecutive segment threshold
G	The connected snapshot threshold
Q	A co-movement
V_t	A candidate co-movement at snapshot S_t
$Q_t{}^*$	A primary candidate co-movement
Q_t^i	A sub-candidate co-movement of $Q_t{}^*$
$\mathbb{Q}_t$	The set of co-movement
$\mathbb{V}_t$	The set of candidate co-movement

Table 1 summarizes the notations used frequently throughout the paper.

3 Real-time Co-movement Mining

3.1 Overview of RTCM

Fig. 2 illustrates the workflow of RTCM, which includes three main phases: (1) global partitioning, (2) object clustering, and (3) co-movement generation. First, the global partitioning strategy dynamically distributes objects uniformly across partitions for each snapshot, ensuring continuous load balancing in streaming scenarios. Second, a distributed DBSCAN algorithm based on the global partitioning is developed to group the objects in the current snapshot S_t, forming the set of snapshot clusters $\mathbb{C}_t$. Third, the clusters in $\mathbb{C}_t$ are compared in parallel with the candidate co-movements from the most recent G snapshots, identifying valid co-movement patterns in the current snapshot S_t.

3.2 Global Partitioning

The global partitioning consists of two phases: the initialization phase and the dynamic adaptive adjustment phase.

In the initialization phase, all trajectory points from the first snapshot of the dataset are collected and sorted by latitude. The trajectories are then sequentially distributed to corresponding partitions based on their latitude order. Given the number of partitions N and the total number of trajectory points in the snapshot $totalObjects$, each partition is assigned approximately $totalObjects/N$ objects, thereby determining the partition boundaries. However, collecting and sorting trajectory points for every snapshot is computationally expensive, even though it ensures partition balance. To address this issue, we propose a dynamic adaptive partitioning adjustment phase that avoids frequent sorting while maintaining partition balance.

In the adaptive adjustment phase, each partition is equipped with a local counter to track the number of objects within the partition. Additionally, a global counter is employed to record the total number of objects in a single snapshot.

The adaptive adjustment strategy operates as follows: Given the partition threshold r, if any partition's object count exceeds the average across all partitions by r%, partition skew is detected. In this case, repartitioning is triggered in the next snapshot to redefine the partition boundaries. This strategy maintains balance by adjusting partition boundaries without increasing their number, thus avoiding extra shuffling. Since data distribution has already been executed for the current skewed snapshot, the adjustment is deferred to the subsequent snapshot to maintain consistency. Fig. 3 shows the advantages of dynamic partitioning, which will adaptively adjust the size of partitions over time to avoid partition skew. Static partitioning is not suitable for streaming scenes, as the number of objects in certain partitions may increase dramatically over time.

Partitioning cost. Dynamic adjustment is only triggered for the next snapshot when partition skew is detected. Due to the strong continuity of trajectory data between consecutive snapshots, the object distribution does not change drastically, so such repartitioning operations occur infrequently. Their overhead is amortized over the many snapshots that require no adjustment, making the average partitioning cost per snapshot negligible.

Clustering cost. The cost of clustering is typically super-linear in the number of points per partition. Therefore, load imbalance drastically increases the processing time of the slowest node, which becomes the bottleneck for overall latency. The key advantage of the partitioning strategy is that it continuously ensures that each worker node processes a nearly equal amount of data, thereby maximizing load balancing and directly optimizing the clustering step, which dominates the total cost.

3.3 Object Clustering

We adopt DBSCAN [8] for object clustering, as it aligns well with real-time co-movement mining requirements by naturally capturing arbitrary-shaped, density-based object aggregations (e.g., irregular traffic jams). While minor irrelevant object clustering may occur in individual snapshots, the co-movement temporal constraint k filters such noise, ensuring results meet the criteria of sustained spatial proximity and consistent movement over consecutive snapshots. Notably, DBSCAN is widely used as the core clustering method in existing co-movement mining works [5, 9, 28, 37]. In contrast, alternative clustering algorithms are less suitable for this scenario. K-type methods [1, 4] require predefined cluster counts, which is impractical for dynamic streaming scenarios with uncertain group numbers per snapshot, and they cannot capture irregular shapes. Density peaks clustering [30] incurs high computational complexity from global density calculations, conflicting with real-time latency requirements, and struggles to identify clusters without distinct density peaks.

Optimized DBSCAN methods [3, 12, 26, 35] are non distributed, these also do not meet real-time latency requirements. Some distributed DBSCAN methods [10, 38] only implement parallelized clustering, while merging between partitions is still sequential. Chen *et al.* [5] propose an online distributed DBSCAN, GR-DBSCAN, using a Grid-RTree index. This method suffers from imbalanced partitioning and inefficient merging. RTCM integrates a grid-based distributed DBSCAN (GD-DBSCAN) for efficient clustering within a single snapshot, which operates on top of our adaptive global partitioning scheme to dynamically maintain load balance across the distributed environment. GD-DBSCAN consists of three steps: (1) **grid data partitioning**, (2) **local DBSCAN clustering**, and (3) **two-level cluster merging**.

To further leverage the advantages of distributed computing, we enhance the global partitioning with an additional grid-based partitioning layer, enabling finer-grained distributed clustering.

Definition 9. ***(Grid cell)*** *A grid g_i is a square region created by imposing a regular grid on the region of a dataset, and i is the grid's index. Moreover, each grid cell has a width of ε.*

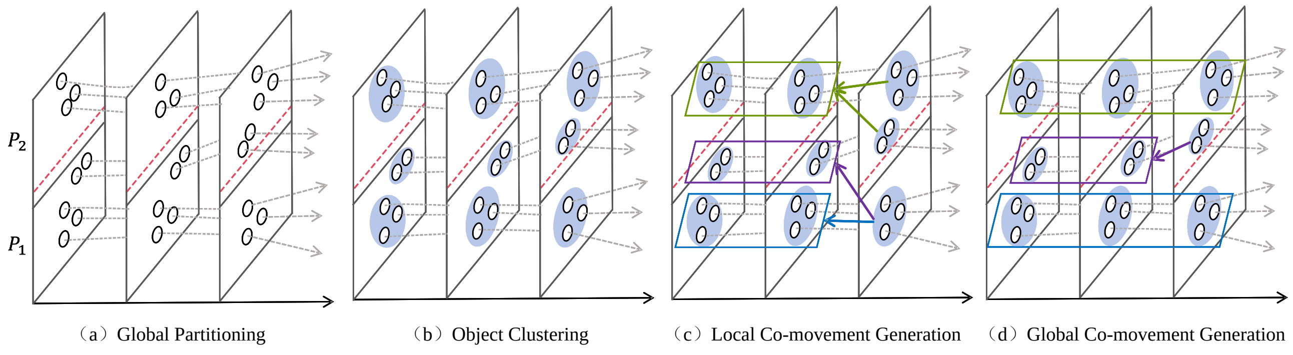
The grid index of an object o is calculated as follows:

$$g_i(o) = \left\lfloor \frac{|o_{lon} - b_{lon}|}{\varepsilon} \right\rfloor + \left(\left\lfloor \frac{|o_{lat} - b_{lat}|}{\varepsilon} \right\rfloor \times GX \right), \quad (1)$$

where b is the left-bottom point in the dataset, $GX = \left\lfloor \frac{|c_{lon} - b_{lon}|}{\varepsilon} \right\rfloor$ is the number of grid columns, and c is the top-right point of the dataset. For example, in Fig. 4(b), with $\varepsilon = 3$, the grid index of object $o_1(4, 10)$ is g_{13}.

A grid partition GP_i is the set of points in a cell g_i and its neighboring cells. For example, in Fig. 4(b), grid partition GP_9 includes points from cells $\{g_4, g_5, g_6, g_8, g_9, g_{10}, g_{12}, g_{13}, g_{14}\}$.

Local DBSCAN clustering. Local DBSCAN applies the traditional DBSCAN [8] for clustering. Using parameters ε and *minPts*, Local DBSCAN runs in parallel across grid partitions. A point p is a core point if $p.neighbors \geq minPts$, where $p.neighbors$ includes all points within a distance ε from p. Local DBSCAN starts by randomly selecting a point p. If p is a core point, its neighbors are



P_2
P_1
（a）Global Partitioning
（b）Object Clustering
（c）Local Co-movement Generation
（d）Global Co-movement Generation

Figure 2: The workflow of RTCM.

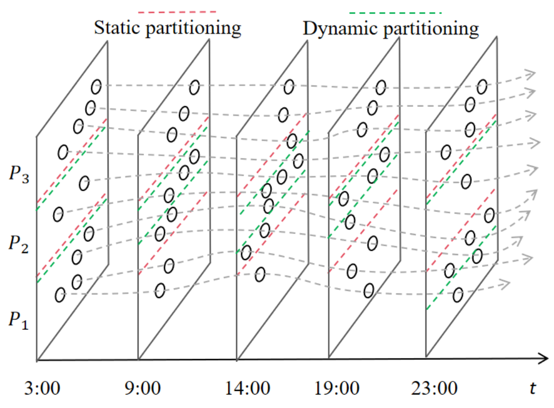


Figure 3: An example of static and dynamic partitioning.

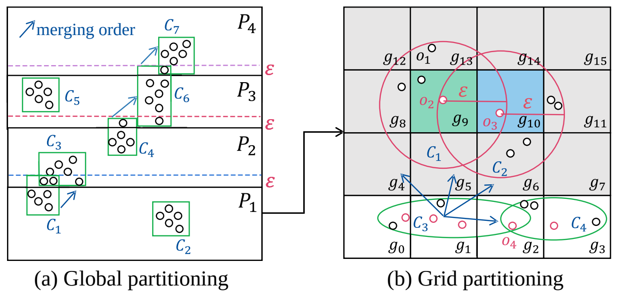


Figure 4: An example of two-level data partitioning.

collected into a cluster C. The algorithm then iteratively adds the neighbors of other core points in C until all points in C are processed. This process is repeated for all points in a local partition to generate clustering results. To improve efficiency, partitions with fewer than *minPts* objects are disregarded, as they cannot form clusters.

Two-level cluster merging. Since clusters from a single local partition are incomplete, it is necessary to perform *local cluster merging*. It merges the clusters obtained from different grid partitions within each global partition in parallel. Clusters are sorted in ascending order by their partition indices and are processed sequentially to ensure orderly merging. The merging follows two principles. (1) If a point p exists in multiple local partitions and is a core point in at least one of them, the corresponding clusters are merged. (2) Each cluster only merges with clusters from neighboring partitions with higher indices.

Example 3. *In Fig. 4(b), let minPts = 5, with core points marked in red. Point o_2 is a core point in cluster C_1 in grid partition GP_9 and also belongs to cluster C_2 in grid partition GP_{10}. As a result, C_1 and C_2 are merged to form a larger cluster. Cluster C_3 is located in partition GP_1, it only merges with clusters in neighboring partitions GP_2, GP_4, GP_5, and GP_6, as indicated by the arrows.*

After merging all local partitions in a global partition, clusters at the global partition boundaries may merge with ones in adjacent partitions to form larger clusters. Boundary clusters are sent to the master node for *global cluster merging*.

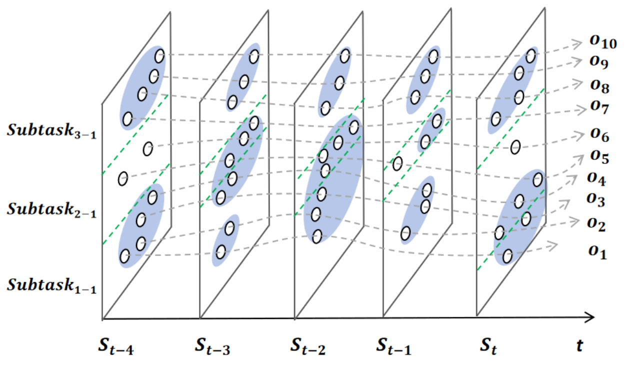
First, clusters from each global partition are arranged by their index, and merging is performed between clusters from adjacent global partitions in sequential order. In Fig. 4(a), the arrows indicate the order of global cluster merging.

The clusters obtained by global cluster merging on the master node will be sent back to the worker nodes for subsequent co-movement generation. Clusters that span multiple global partitions will be sent to the worker nodes with smaller indices.

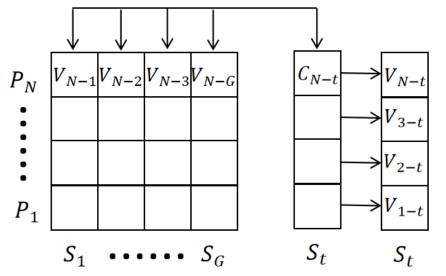
Correctness of core points. The completeness of core points ensures that no core points and their ε-neighborhood relationships are missed during merging. Each grid partition GP_i includes a central cell and its neighboring cells. For a core point p located in the central cell of GP_i, all its neighboring points within a distance of ε are guaranteed to be in GP_i, which covers p's ε-neighborhood. This ensures that no loss of core points. For any pair of core points with a distance less than ε, both points will belong to the same local partition. As a result, during the merging of clusters across grid partitions, all core points within a distance of ε are clustered together, ensuring that the final clustering result is correct.

We analyze the time complexity of each step of GD-DBSCAN. Both global and local partitioning have complexity $O(1)$. Local DBSCAN clustering has complexity $O((\alpha/\beta)\cdot\gamma^2)$, where α is the number of valid local partitions, β is the degree of parallelism, and γ is the number of objects in a local partition, with $O(\gamma^2)$ being the complexity of DBSCAN on a single partition. Local cluster merging takes $O((\alpha/N)\cdot\gamma)$ time, where N is the number of global partitions. Global cluster merging has complexity $O(n_c \cdot n_o)$, where n_c is the number of clusters at global partition boundaries and n_o is the number of objects per cluster. Hence, the time complexity of GD-DBSCAN is $O((\alpha/\beta)\cdot\gamma^2 + (\alpha/N)\cdot\gamma + n_c \cdot n_o)$

Novelty of GD-DBSCAN. GD-DBSCAN has three novel features. First, the global partitioning ensures balanced load distribution across all worker nodes, effectively mitigating the quadratic



(a) The partitioning of subtasks.



(b) The co-movement generation of subtasks.

Figure 5: The subtasks of RTCM.

computational cost of DBSCAN by maintaining approximately equal-sized point sets on each node. Second, the partition boundary design guarantees spatially contiguous grid indexes within individual nodes. This allows local clustering to be fully completed on each node before cross-node cluster merging is initiated. Most clusters are thus confined within local partitions, drastically reducing inter-node data transmission overhead. Finally, the two-level cluster merging strategy ensures the correctness of both cross-grid cluster merging and cross-node cluster merging.

3.4 Real-time Co-movement Generation

Real-time co-movement generation aggregates clusters in S_t with candidate co-movements obtained from previous G snapshots $(S_{t-G}, ..., S_{t-1})$ to produce candidate co-movements for S_t. Specifically, it divides G partitions on the basis of global partitions according to snapshot granularity, with each snapshot serving as a snapshot-level partition. Each parallel subtask is determined by time and spatial dimensions. The time dimension is defined by the snapshot, while the spatial dimension is determined by the global partition. Intersection the candidate co-movements in each subtask with the clusters in snapshot S_t to mine valid co-movements, and real-time output the co-movements obtained in all subtask. Fig 5a shows the partitioning of subtasks. The temporal dimension consists of 4 snapshots, and the spatial dimension contains 3 partitions, thus forming $3 * 4$ parallel subtasks, where each subtask includes the candidate co-movements defined by its corresponding spatio-temporal partition, as enclosed by the blue circles.

It consists of two stages: **local co-movement generation** and **global co-movement generation**. Local co-movement generation mines co-movements by aggregating candidate co-movements in each subtask and clusters in S_t. A small number of special candidate co-movements and clusters that have not been aggregated on the subtask are sent to the master node for global co-movement generation. Since local and global co-movement generation follow similar strategies, we describe only the local co-movement generation and highlight the differences for brevity. Fig 5b shows the local co-movement generation in each subtask. Each grid cell represents a subtask that contains a set of candidate co-movements. Within each subtask, candidate co-movements are aggregated with existing clusters to generate qualified co-movements at snapshot S_t.

Candidate co-movement mining. The dynamic arrival and departure of objects creates an inclusion relationship on the object sets of co-movements. Based on this inclusion relationship, we design a novel structure of candidate co-movements. A candidate co-movement V_t includes one primary candidate co-movement $Q_t{}^*$ and several sub-candidate co-movements $\langle Q_t^1, Q_t^2, \ldots, Q_t^h \rangle$ ordered by their starting time in ascending order.

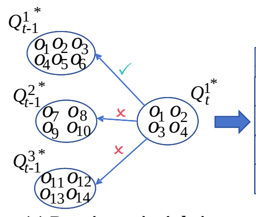
A brute-force enumeration scheme for obtaining $\mathbb{V}_t$ is to compare each cluster C_t in $\mathbb{C}_t$ with each candidate co-movement V_{t-1} in $\mathbb{V}_{t-1}$. If $|V_{t-1}.O \cap C_t.O| \geq m$, V_{t-1} can be merged with C_t to form a candidate co-movement at S_t. This brute-force enumeration is inefficient as it involves numerous unnecessary intersections. To address this, we propose two efficient candidate co-movement pruning strategies for each global partition to minimize intersection operations between snapshots.

Primary candidate co-movement pruning. According to the definition of primary candidate co-movements, we first initialize each cluster $C_t^i \in \mathbb{C}_t$ in the current snapshot as a primary candidate co-movement Q_t^{i*}. After initialization, we obtain an incomplete set of candidate co-movements $\mathbb{V}_t = \{V_t^1, V_t^2, \ldots, V_t^n\}$, where V_t^i only includes the primary candidate co-movement Q_t^{i*}. Then, we intersect $V_t^i.Q_t^{i*}$ with the candidate co-movements in V_{t-1}^j to obtain $V_t^i.SubQ_t^i$. Due to $SubQ_{t-1}^j.O$ is a subset of $Q_{t-1}^j{}^*.O$, we propose a novel primary candidate co-movement pruning strategy (PCCP).

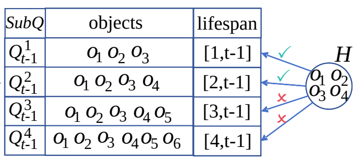
The pruning principles for PCCP are as follows. (i) If the intersection result H between $Q_t^{i*}.O$ and $Q_{t-1}^j{}^*.O$ has less than m objects, intersection operations between Q_t^{i*} and $SubQ_{t-1}^j$ can be skipped. This is because the intersection between Q_t^{i*} and any candidate co-movements in $SubQ_{t-1}^j$ has fewer than m objects and fails the cardinality condition to form a co-movement. (ii) If a sub-candidate co-movement Q_{t-1}^k in $SubQ_{t-1}^j$ has an object set identical to H, intersections of Q_t^{i*} with subsequent sub-candidate co-movements in $SubQ_{t-1}^j$ can be skipped. These intersections would yield at most H, but with a shorter lifespan than Q_{t-1}^k.

Example 4. *Fig. 6 exemplifies PCCP. In particular, Fig. 6(a) illustrates pruning principle i. The intersections between $Q_t^{1*}.O$ and $\{Q_{t-1}^2{}^*.O, Q_{t-1}^3{}^*.O\}$ have fewer than m objects; it is unnecessary to intersect $Q_t^{1*}.O$ with the sub-candidate co-movements of $\{Q_{t-1}^2{}^*.O, Q_{t-1}^3{}^*.O\}$. Fig. 6(b) illustrates pruning principle ii. Here, sub-candidate co-movements of $Q_{t-1}^1{}^*$ are ordered by their starting time in ascending order. The intersection H between $Q_t^{1*}.O$ and $Q_{t-1}^1{}^*.O$ is identical to the sub-candidate co-movement $Q_{t-1}^2.O$; thus, the intersection with the remaining candidate co-movements can be skipped.*

Sub-candidate co-movement pruning. Two observations apply to co-movements. First, an object can only be located in one primary candidate in a snapshot. Second, a cluster does not change dramatically from one snapshot to the next. Based on these observations, we propose a novel sub-candidate co-movement pruning strategy (SCCP). SCCP works as follows: a



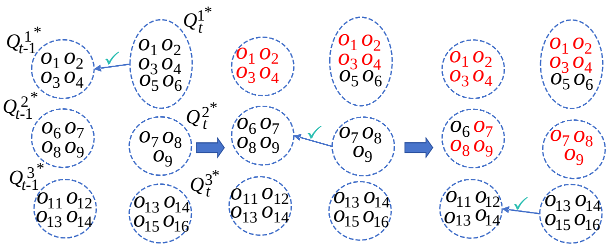
(a) Pruning principle i



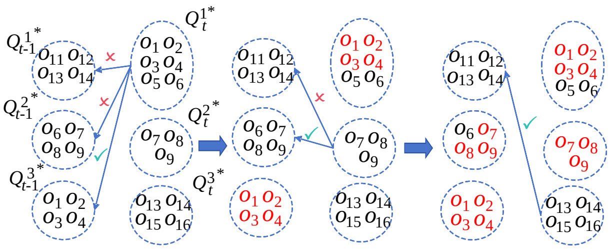
SubQ	objects	lifespan
$Q^1_{t\text{-}1}$	$o_1\ o_2\ o_3$	[1,t-1]
$Q^2_{t\text{-}1}$	$o_1\ o_2\ o_3\ o_4$	[2,t-1]
$Q^3_{t\text{-}1}$	$o_1\ o_2\ o_3\ o_4\ o_5$	[3,t-1]
$Q^4_{t\text{-}1}$	$o_1\ o_2\ o_3\ o_4 o_5\ o_6$	[4,t-1]

(b) Pruning principle ii

Figure 6: An example of PCCP.



(a) The best case.



(b) The worst case.

Figure 7: An example of SCCP.

counter is employed for each primary candidate to record the number of objects that have already intersected with others. If the size of a primary candidate co-movement minus the counter is less than m, it is unnecessary to intersect with the remaining primary candidate co-movements.

Example 5. *Fig. 7 illustrates SCCP. The red objects denote the intersections of the primary candidate co-movements. Given $m = 3$, we need nine intersection operations for pairwise comparisons between $\mathbb{V}_t$ and $\mathbb{V}_{t-1}$ without SCCP. With SCCP, only three intersection operations are required in the best case, e.g., $\{(Q^{1*}_t \cap Q^{1}_{t-1}{}^{*}), (Q^{2*}_t \cap Q^{2}_{t-1}{}^{*}), (Q^{3*}_t \cap Q^{3}_{t-1}{}^{*})\}$. After the first round of intersections, if the number of remaining objects in Q^{1*}_t falls below m, it is excluded from further intersections with other primary candidate co-movements in $\mathbb{V}_{t-1}$. In the worst case, up to six intersections are needed, e.g., $\{(Q^{1*}_t \cap Q^{3}_{t-1}{}^{*}, Q^{2}_{t-1}{}^{*}, Q^{1}_{t-1}{}^{*}), (Q^{2*}_t \cap Q^{3}_{t-1}{}^{*}, Q^{2}_{t-1}{}^{*}), (Q^{3*}_t \cap Q^{3}_{t-1}{}^{*})\}$.*

Candidate co-movements in one global partition may intersect with clusters from other global partition, caused by groups of objects moving across global partitions. To address this, candidate co-movements and clusters that do not satisfy SCCP are sent to the master node for global co-movement generation through the side output stream. Since few objects typically move between partitions in adjacent snapshots, the data for global co-movement generation is limited. The pruning strategies can also be applied analogously.

Existing method [5] treat all cluster subsets of size no less than m as candidate co-movements, which is redundant under the same lifespan constraint. For instance, it is unnecessary to generate the subset co-movement $< \{o_1, o_2\}, [t_1, t_5] >$ when the complete cluster $< \{o_1, o_2, o_3\}, [t_1, t_5] >$ already exists. In contrast, RTCM only performs intersection operations on clusters themselves and integrates dedicated pruning strategies to eliminate redundant subset generation, thus it has less candidate co-movements.

In Algorithm 1, we provide the pseudo-code implementation of real-time co-movement mining via candidate co-movement pruning. First, it initializes a cluster C^i_t at the current snapshot as a primary candidate co-movement Q^{i*}_t (line 2). Next, it performs PCCP between $V^i_t \in \mathbb{V}_t$ and $V^j_{t-1} \in \mathbb{V}_{t-1}$ to obtain $V^i_t.SubQ^i_t$ (lines 3–5). If the number of objects in $Q^{i*}_t.O$ minus the counter is less than m, V^i_t is skipped (lines 13–15). Finally, we output the real co-movements with $lifespan \geq k$ and $T_e \geq L$ from $\mathbb{V}_t$ in each worker node (lines 18–19), where T_e is the end time segment of co-movement.

We analyze the time complexity of RTCM. In the best case, SCCP has complexity $O(|\mathbb{V}_{t-i}|)$. In the worst case, SCCP has complexity $O(|\mathbb{V}_{t-i}| \cdot |\mathbb{V}_t|)$. In the average case, the expected pruning time occurs at one-half, leading to an overall time complexity of $O(|\mathbb{V}_{t-i}| \cdot |\mathbb{V}_t|)$, where $|\mathbb{V}_{t-i}|$ is the number of candidate co-movements and $|\mathbb{C}_t|$ is the number of clusters. Under the effect of PCCP, $\mathbb{V}_t$ is merged only with primary candidate co-movements whose intersection size exceeds m, resulting in a time complexity of $O(|V_{t-i}|)$, where $|V_{t-i}|$ is the number of sub-candidate co-movements. Hence, the time complexity of RTCM is $O(|\mathbb{V}_{t-i}| \cdot |\mathbb{C}_t| \cdot |V_{t-i}|)$.

Subsequence-based set intersection. PCCP and SCCP reduce the number of intersection operations between snapshots substantially. However, it still involves many cluster intersection

Algorithm 1 Real-time Co-Movement Mining

Input: a set of candidate co-movements $\mathbb{V}_{t-i}$, a set of snapshot clusters $\mathbb{C}_t$, a duration threshold k, a cardinality threshold m, a consecutive segment threshold L

Output: The result set of real co-movement $\mathbb{Q}_t$

```
1:  $\mathbb{V}_t \leftarrow \emptyset, \mathbb{Q}_t \leftarrow \emptyset$
2:  Initialize $\mathbb{C}_t$ to $\mathbb{V}_t$
3:  for $V^i_t \in \mathbb{V}_t$ do
4:      for $V^j_{t-i} \in \mathbb{V}_{t-i}$ do
5:          $V^i_t.SubQ^i_t$ add $PCCP(V^i_t, V^j_{t-i})$
6:          $S \leftarrow Intersection(Q^{i*}_t.O, Q^{j}_{t-1}{}^{*}.O)$
7:          $Q^{i*}_t.counter += S.size,$
8:          $Q^{j}_{t-i}{}^{*}.counter += S.size$
9:          if $Q^{j}_{t-1}{}^{*}.size - Q^{j}_{t-1}{}^{*}.counter < m$ then
10:             Label $V^j_{t-1}$ as intersected        /* Skip $V^j_{t-i}$ */
11:         end if
12:     end for
13:     if $Q^{i*}_t.size - Q^{i*}_t.counter < m$ then
14:         Label $V^i_t$ as intersected        /* Skip $V^i_t$ */
15:     end if
16: end for
17: Send $\mathbb{V}_t$ and $\mathbb{V}_{t-i}$ that are not intersected to master node
18: Add $\mathbb{V}_t.Q^j_t$ to $\mathbb{Q}_t$, if $Q^j_t.lifespan \geq k$ and $Q^j_t.T_e \geq L$
19: return $\mathbb{Q}_t$
```

computations. A cluster may contain two objects with significantly different ids. Therefore, the span between two adjacent object ids in a cluster may be relatively large. Based on this observation, we employ a subsequence-based set intersection (SSI) method to improve the efficiency of set intersections.

SSI first sorts two intersecting sets by object ids. It then divides the sorted sets into multiple smaller subsequences and performs intersection operations for subsequences whose elements fall in the same range, where the size of each subsequence is $\left\lfloor \sqrt{\text{set.size}} \right\rfloor$. Before intersecting two subsequences, SSI first compares the start and last elements of the two subsequences. If the last element of a subsequence is smaller than the start element of the other, SSI skips the intersection between them.

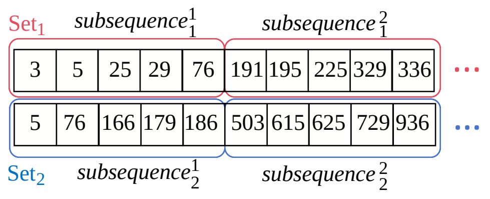


Figure 8: An example of subsequence-based set intersection.

Example 6. *Fig. 8 shows the division of two sets. For practical reasons, only the first two subsequences of each set are shown in the figure, each with a size of 5. The intersection results between subsequence$_1^1$ and subsequence$_2^1$ is $\{5, 76\}$, and the intersection between subsequence$_2^1$ and subsequence$_1^2$ is skipped.*

4 Experimental Study

We evaluate the performance of RTCM on a cluster with nine nodes, with one serving as a master node and eight as worker nodes. Each node is equipped with a 16-core Intel Cooper Lake processor and 64GB of RAM. The cluster runs HDFS 2.8.5 and Flink 1.14.5. HDFS is used for distributed storage of the datasets, and the Flink computing engine reads data from HDFS, which realistically simulates the distributed data collection process in practical real-time scenarios.

4.1 Experimental Settings

Datasets. We use four real-world datasets to model streaming trajectories. All datasets are summarized in Table 2. The sampling interval is determined by the data collection device.

- **GeoLife** contains trajectories of 182 users over three years, collected through GPS devices. Approximately 91% of the trajectories are sampled every 5 seconds. Each user's data on different dates is treated as a separate trajectory, increasing the number of trajectories from 182 to 17,621.
- **IMIS** captures the trajectories of ships in the Eastern Mediterranean during one month. The dataset is collected using the Automatic Identification System (AIS), which tracks vessels.
- **AIS** is obtained through the U.S. Coast Guard's onboard navigation safety devices, which monitor the location and characteristics of vessels. The dataset represents 10 days of vessel activity in U.S. and international waters.
- **Chengdu** consists of taxi trajectory data collected in Chengdu over a 10-day period from November 1 to 10, 2016. Each trajectory corresponds to the travel path of an individual taxi during a ride-hailing service.

Table 2: Datasets (M: million, s: second).

Property	GeoLife	IMIS	AIS	Chengdu
# of Trajectories	17,621	101,679	457,536	2,025,032
# of Points	24.9M	11.8M	120M	322M
# of Snapshots	17,280	259,200	14,400	288,000
Sampling Interval (s)	5	10	60	3
Points per Snapshot	1,727	454	8,365	1,117

Baselines. We compare with three state-of-the-art real-time algorithms for co-movement mining.

- **FBA** [5] is a real-time distributed co-movement pattern mining algorithm based on sliding windows, adopting fixed length bit compression with low latency.
- **VBA** [5] is also a real-time distributed co-movement pattern mining algorithm based on sliding windows, adopting variable length bit compression with high throughput.
- **ECS** [34] is a centralized graph-based algorithm for real-time co-movement pattern mining. It identifies co-movement behaviors by monitoring cluster activities.

Parameters. Table 3 shows the parameters of our method. ε is set as a percentage of the dataset's maximum extent. Since both ε and *minPts* are used to control density-based clustering, ε is more sensitive to the clustering results. Therefore, we vary only ε, with *minPts* fixed at 10[5, 9, 28, 37]. In each experiments, we vary one parameter while fixing the others at their default values. We set the degree of parallelism $\beta = 8$ for all methods.

Table 3: Default parameter settings.

Parameters	Descriptions	Default values
ε	Search radius of DBSCAN	0.04%
m	The significant threshold	15
k	The consecutive threshold	180
N	The number of worker nodes	8
L	The segment threshold	60
G	The connected threshold	15
r	The partition threshold	20%

Performance metrics. We use **latency** and **throughput** to assess efficiency. The average processing time for each snapshot is reported as the latency, while the throughput is reported as the number of snapshots processed per second. The latency is calculated as follows:

$$Latency = \frac{TotalResponseTime}{SnapshotsCount}, \tag{2}$$

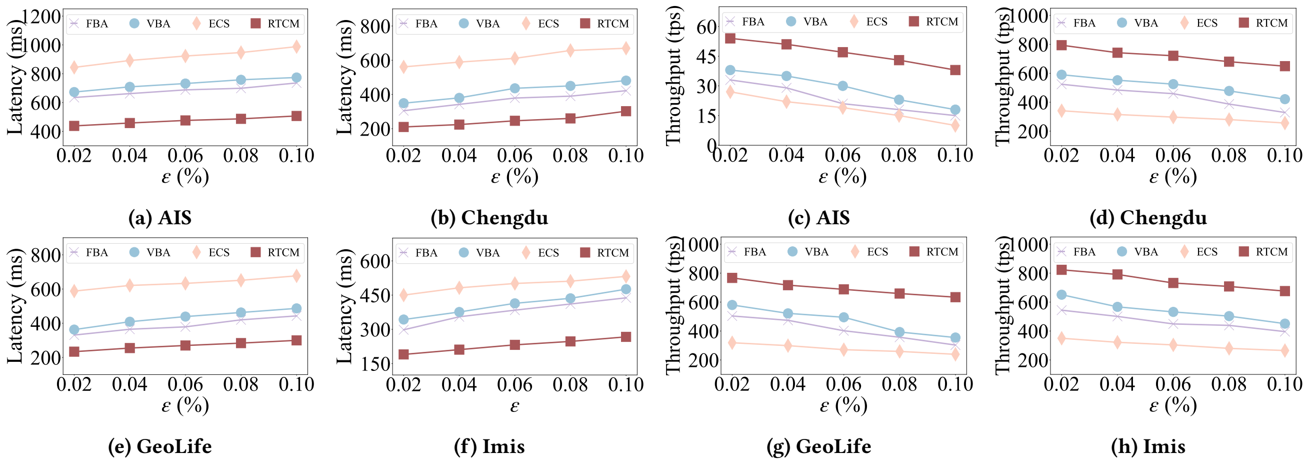
where *TotalResponseTime* is the sum of all *ResponseTime* values, and *SnapshotsCount* is the total number of processed snapshots. The *ResponseTime* of a snapshot is the time from the occurrence of a snapshot to the completion of its processing.

The throughput is calculated as follows:

$$Throughput = \frac{SnapshotsCount}{ExecuteTime}, \tag{3}$$

where the *ExecuteTime* is the time from the input of the first snapshot to the completion of the processing of the last snapshot, and *SnapshotsCount* is the number of all processed snapshots.

Specifically, latency refers to the elapsed time from the input of a spatio-temporal snapshot to the output of its co-movement mining results, which directly determines whether the system



(a) AIS (b) Chengdu (c) AIS (d) Chengdu
(e) GeoLife (f) Imis (g) GeoLife (h) Imis

Figure 9: Impact of parameter ε.

meets real-time requirements by ensuring processing completion within the interval between consecutive snapshots. Throughput denotes the volume of trajectory data points processed per unit time, reflecting the system's capacity to handle large-scale streaming data.

We adopt the Jaccard similarity [6] to quantify the consistency between RTCM's co-movements and those of baselines. The Jaccard similarity is defined as follows:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}, \tag{4}$$

where $|A \cap B|$ denotes the number of common co-movement patterns and $|A \cup B|$ denotes the total number of unique co-movement patterns between RTCM's co-movements and those of baselines.

4.2 Efficiency and Parameter Study

Fig. 9 – 13 report the latency and throughput of all methods under different parameter settings. *tps* is the number of snapshots processed per second. It is shown that RTCM outperforms FBA and VBA by approximately 50% in both latency and throughput. There are three main reasons. First, RTCM processes trajectories based on snapshots, avoiding the processing of duplicate subtrajectories between sliding windows. Second, RTCM optimizes DBSCAN to accelerate the clustering stage. Third, RTCM adopts two efficient pruning strategies and optimizes set intersection to accelerate co-movement generation. Due to space limitations, we present all the results of Impact of ε in Fig. 9 and the results of other parameters for AIS and Chengdu in Fig. 10 – 13.

Impact of ε. Figs. 9a–9h show the change in the latency and throughput as ε increases. As expected, the performance drops as ε increases. This is because the average cluster size increases as ε increases, leading to higher time costs in the clustering stage.

Impact of m. Figs. 10a–10d show the change in the latency and throughput as m increases. The performance increases as m increases. This is because larger m requires that more objects to move together to form a co-movement, reducing the number of candidate co-movements in each snapshot for RTCM. FBA and VBA enumerate all object sets in a sliding window with a cardinality of at least m. Larger m cause fewer enumerations, thus improving the efficiency of co-movement generation.

Impact of k. Figs. 11a–11d show the change in the latency and throughput as k increases. The performance increases with the increase of k. For RTCM, larger k requires objects to move together for a longer period of time, reducing the number of co-movements in each snapshot. For FBA and VBA, larger k leads to fewer enumeration results within the window that meet the lifespan condition, reducing the number of enumerations.

Impact of L. Figs. 12a–12d show the change in the latency and throughput as L increases. The performance increases with the increase of L. For RTCM, a larger L increases the threshold for time segments, reduces the number of candidate co-movements in snapshots. It also reduces the amount of data processed by parallel subtasks, thus improving efficiency. For FBA and VBA, like k, larger L leads to fewer enumeration results within the window that meet the lifespan condition, reducing the number of enumerations.

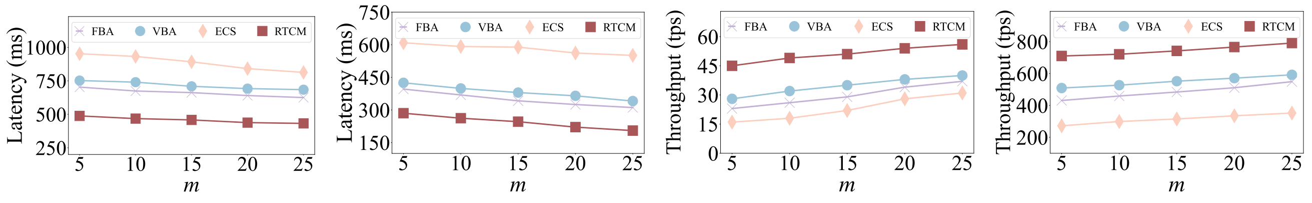
Impact of G. Figs. 13a–13d show the change in the latency and throughput as G increases. The performance decreases with the increase of G. For RTCM, as G increases, the number of parallel subtasks increases and the number of candidate co-movements increases, resulting in a decrease in performance. For FBA and VBA, larger G leads to more enumeration results within the window that meet the connected condition, increasing the number of enumerations.

4.3 Scalability

We evaluate the scalability by varying the number of worker nodes N and the number of objects O_r, where O_r represents the ratio of objects in the dataset. We present the latency and throughput results on AIS and Chengdu in Fig. 14.

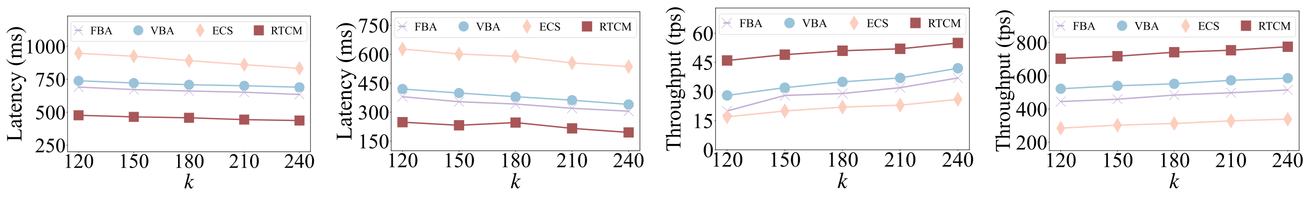
Impact of N. Figs. 14a–14c present the changes in latency and throughput as the number of worker nodes N increases. As expected, performance improves as N increases, due to the increased computational capacity. However, the improvements decrease due to two main reasons. First, the datasets are relatively small, and adding more nodes does not enhance performance markedly. Second, adding more nodes incurs higher network costs and additional overheads from creating and managing resources across nodes.

Impact of O_r. Figs. 14e–14g show that RTCM outperforms all baselines in terms of latency and throughput. As the number of objects (O_r) increases, the performance decreases gradually. This is because a fixed number of nodes results in a higher object load



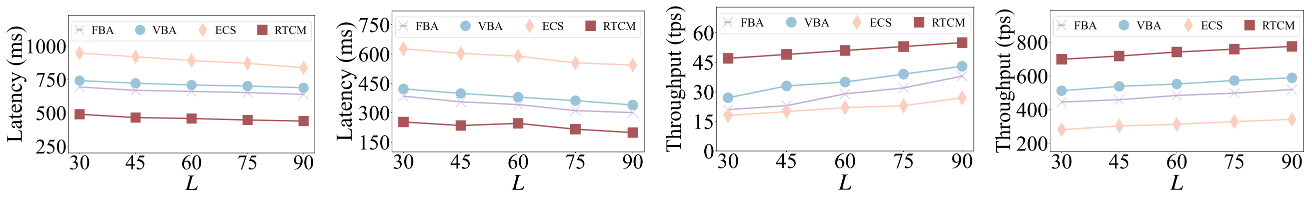
(a) AIS (b) Chengdu (c) AIS (d) Chengdu

Figure 10: Impact of parameter m.



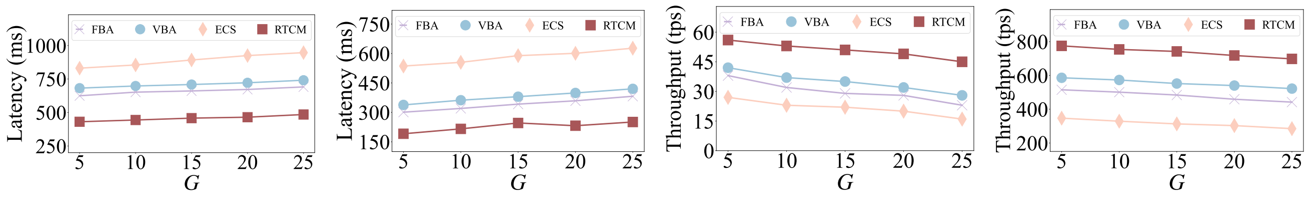
(a) AIS (b) Chengdu (c) AIS (d) Chengdu

Figure 11: Impact of parameter k.



(a) AIS (b) Chengdu (c) AIS (d) Chengdu

Figure 12: Impact of parameter L.



(a) AIS (b) Chengdu (c) AIS (d) Chengdu

Figure 13: Impact of parameter G.

per node and per snapshot, increasing computational loads. However, the performance suggest that RTCM scalability and efficiency in managing large-scale data.

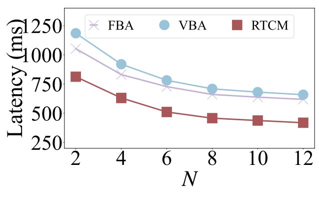
4.4 Efficiency of Dynamic Partitioning

We compare the latency of dynamic partitioning and static partitioning, and conduct a sensitivity analysis of the partitioning threshold r. Fig. 15 shows that the performance of dynamic partitioning is superior to that of static partitioning. Specifically, the dynamic strategy can real-time adapt to the density of streaming trajectory data, maintain load balancing among distributed worker nodes, and thus effectively reduce the computational overhead of local clustering and inter-node data transmission consumption during the GD-DBSCAN clustering stage.

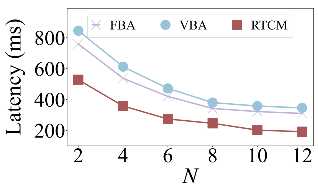
Impact of r. As the partition threshold r increases, the performance of dynamic partitioning first improves and then declines. Dynamic partitioning achieves optimal performance when r is set to 20%. At this value, the frequency of partition skew detection is low, and the slight load imbalance among worker nodes will not cause significant performance degradation in the clustering phase. In contrast, when $r = 10\%$, frequent partition skew is detected, triggering excessive dynamic partitioning processes and thus leading to performance loss. When $r \geq 30\%$, severe partition skew occurs among worker nodes, which consequently deteriorates the performance of the clustering phase.

4.5 Efficiency of GD-DBSCAN

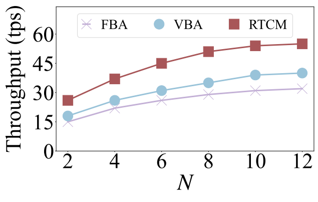
We compare the latency and throughput of GD-DBSCAN with those of GR-DBSCAN[5] and the traditional DBSCAN [8] during the clustering stage only. GR-DBSCAN is a distributed algorithm based on grid partitioning that iterates through all object pairs with distances less than ε in each partition and then collects



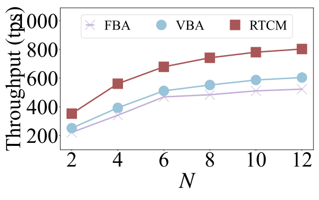
(a) AIS



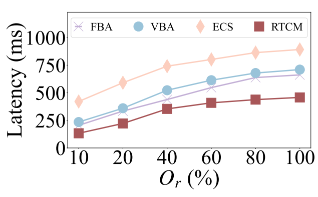
(b) Chengdu



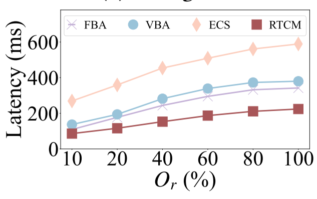
(c) AIS



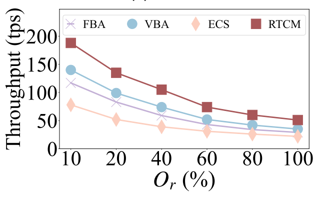
(d) Chengdu



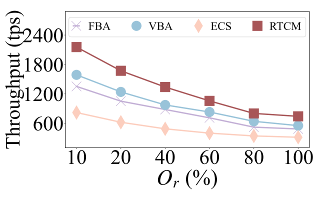
(e) AIS



(f) Chengdu

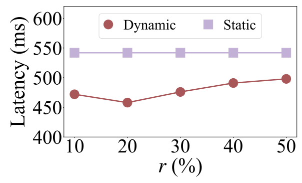


(g) AIS

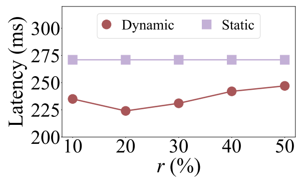


(h) Chengdu

Figure 14: Scalability.

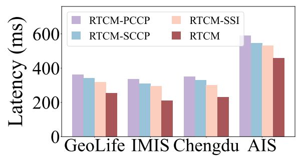


(a) AIS

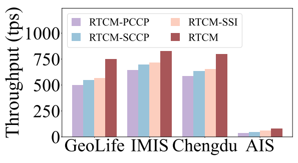


(b) Chengdu

Figure 15: Performance of dynamic partitioning.

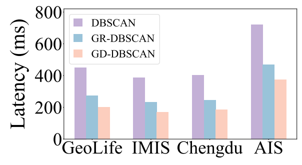


(a) Latency

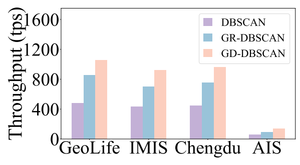


(b) Throughput

Figure 19: Effects of cluster pruning and set intersections.



(a) Latency



(b) Throughput

Figure 16: Performance of different variants of DBSCAN.

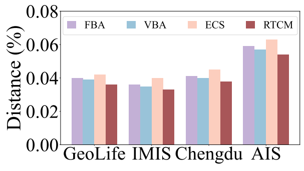


Figure 17: Quality of generated co-movements.

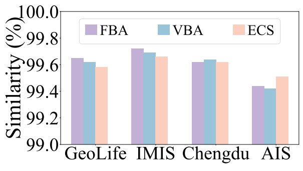


Figure 18: Correctness of generated co-movements.

object pairs from all partitions to form clustering results. FBA and VBA use GR-DBSCAN for object clustering.

Fig. 16 shows that GD-DBSCAN has lower latency and higher throughput than GR-DBSCAN and DBSCAN. GD-DBSCAN processes data by partitioning a single snapshot, clustering each partition independently, and merges the results efficiently. In contrast, GR-DBSCAN, despite being distributed, suffers from inefficiencies because it merges numerous object pairs during the clustering phase and outputs the result pairs for subsequent processing, which reduces performance. DBSCAN, being non-distributed and operating on a single node, is inherently slower and less efficient.

4.6 Quality of Generated Co-movements

We use **Hausdorff distance** [2] to evaluate the quality of generated co-movements. It measures the maximum distance between the closest points of two trajectories. A smaller Hausdorff distance indicates that the trajectories in a co-movement are denser and more closely aligned, indicating a higher-quality co-movement. The Hausdorff distance is normalized by dividing it by the dataset's maximum extent yielding a "distance" percentage.

Figure 17 demonstrates that RTCM produces higher-quality co-movements than the baseline methods. Although VBA, FBA, and RTCM rely on DBSCAN for clustering, VBA and FBA generate subsets of companion objects as co-movements, some of which fail to meet the DBSCAN conditions, such as subsets with objects at opposite extremes of a group. This reduces the average similarity of their co-movements. ECS, on the other hand, clusters objects using connected subgraphs, linking all objects within a distance of ε. This creates larger clusters than with DBSCAN, resulting in lower co-movement quality.

4.7 Correctness of Generated Co-movements

We validated correctness using established baselines as the reference standard, which is a recognized methodology in the field. We quantitatively measured the consistency between RTCM and each baseline using Jaccard similarity. Figure 18 shows that the Jaccard similarity between the co-movement patterns mined by RTCM and those of the baselines exceeds 99%. The minimal discrepancies are attributed to the uncertainty in the selection order of core points during the clustering phase, which leads to the differential assignment of a small number of edge points in clusters and thus causes minor deviations in the co-movement patterns.

4.8 Effects of Cluster Pruning and Intersections

We conduct ablation studies to assess the effects on efficiency of the two pruning strategies and subsequence-based set intersection. Specifically, RTCM-PCCP removes the PCCP pruning strategy during co-movement generation, and RTCM-SCCP removes the SCCP pruning strategy. RTCM-SSI uses the common set intersection method instead of the improved one. Fig. 19 reports the latency and throughput of the four variants. As expected, RTCM has higher throughput and lower latency than RTCM-PCCP, RTCM-SCCP, and RTCM-SSI, indicating that the two pruning strategies reduce the number of intersection operations between snapshots and improve the performance of real-time co-movement mining. Moreover, subsequence-based set intersection accelerates the intersection of two sets.

5 Related Work

5.1 Co-movement Pattern

Several well-defined co-movement patterns exist: flock [11, 36], convoy [16, 17], swarm [22], group [39], and platoon [21]. These patterns impose different constraints on spatial proximity and temporal continuity. Flock and group require that all objects in a cluster are within a radius of r. Swarm, platoon, and convoy use density-based spatial clustering. In the temporal dimension, flock requires that all timestamps for detected clusters are contiguous, which is called global continuity [11]. Swarm does not impose any restrictions on time continuity. Group and platoon adopt a compromise that allows for arbitrary gaps between successive segments, which is known as local continuity [39]. They introduce a parameter L to control the minimum length of local continuous segments. To relax the size and shape constraints required by the flock pattern, Jeung *et al.* [17] proposed the convoy pattern. Convoys satisfy a co-movement pattern where at least m objects move together during at least k consecutive timestamps. Unlike the flock pattern, objects in a convoy are required to be density-clustered, rather than within a predefined shape and with fixed sizes.

5.2 Offline Co-movement Pattern Mining

Many sequential algorithms have been proposed for offline co-movement mining. The sequential algorithms CuTS, CuTS+, and CuTS* [17] attempt to reduce the cost of DBSCAN by filtering of objects. CMC [17] finds convoys by clustering objects at each time snapshot and then extracts the common objects in consecutive time snapshots. Yoon *et al.* [40] observe that CMC may miss larger convoys and may retrieve invalid convoys when the density connectivity among objects is not completely satisfied. To address this issue, they proposed VCoDA to discover partially connected convoys and verify their density connectivity.

Sequential algorithms lack scalability to large datasets. For example, CuTs [17], requires 100 seconds to process a small dataset of 13 cattle over a few hours. To address this, Several parallel algorithms [9, 28, 37] have been proposed. Orakzai *et al.* [28] propose DCM, a distributed convoy mining algorithm, which partitions data by timestamps and processes multiple timestamps simultaneously. However, DCM's performance is highly dependent on partition sizes, and improper settings can make it perform worse than sequential algorithms. Moreover, DCM is unsuitable for streaming processing. SWISP [37] employs sliding window indexing and sub-track partitioning to reduce candidate sub-tracks but supports only offline data. It filters sub-tracks with fewer than k consecutive snapshots, reducing candidates for convoy mining, and addresses data skewness by leveraging temporal and spatial information for partitioning. SPARE [9], a Spark-based framework, implements two parallel approaches on Apache Spark: Star Partitioning and Prior Enumeration. While SPARE lacks native support for streaming, it can be adapted for snapshot-based real-time co-movement mining using Flink. Due to the highly inefficient key-value clustering stage of SPARE, it cannot run on large datasets, so we did not compare with it.

5.3 Real-time Co-movement Pattern Mining

With the increasing demand for online trajectory analysis, real-time streaming trajectory mining has gained attention. Tang *et al.*[33] develop a smart-and-closed discovery algorithm to find companions efficiently in streaming trajectories. Tritsarolis *et al.*[34] propose EvolvingClusters, a graph-based algorithm for real-time co-movement mining, using Maximal Cliques (MC) and Maximal Connected Subgraphs (MCS) to identify patterns like flocks and convoys. However, these methods are limited to single-machine processing and cannot handle large-scale data. Chen *et al.* [5] introduce ICPE, a distributed framework for real-time co-movement mining using sliding windows. ICPE divides trajectories into sequences of consecutive snapshots with a sliding window and mines co-movement patterns within it. It comprises a clustering stage and a pattern enumeration stage. During the clustering stage, objects in a snapshot are clustered using Range-Join and DBSCAN. The DBSCAN algorithm outputs clustering results in the form of id-id pairs for the subsequent enumeration stage. A cluster is split into many id-id pairs, seriously increasing the storage cost of the clustering stage. During the pattern enumeration stage, it uses FBA and VBA for sliding window-based mining, but suffers from redundant trajectory fragments due to overlapping windows, leading to high computational costs.

6 Conclusion

We propose RTCM, a snapshot-based distributed framework for real-time co-movement mining. RTCM enhances efficiency by comparing clusters from consecutive snapshots in parallel subtasks, avoiding redundant sub-trajectory comparisons. It proposes a dynamic and adaptive partitioning strategy tailored for streaming scenarios, and features GD-DBSCAN for parallel clustering with a grid partitioning and merging design, proven correct in distributed environments. Two pruning strategies and a subsequence-based set intersection method further reduce unnecessary computations, accelerating co-movement generation. Experiments on four real-world datasets show that RTCM is typically able to improve latency and throughput by some 50% over state-of-the-art methods while producing higher-quality co-movements.

In future research, it is of interest to extend RTCM to enable real-time prediction of future co-movements, providing insights for applications such as traffic management and logistics optimization.

Acknowledgments

The research presented in this paper is supported in part by the National Natural Science Foundation of China (No. 62272379, T2341003), the Natural Science Basic Research Program of Shaanxi Province (2025JC-JCQN-081), and the XJTU Research Fund for AI Science (2025YXYC004).

7 Artifacts

Real-time co-movement mining is implemented through Flink. See section 4 for experimental settings. The source code is available at https://github.com/XNetLab/RealTimeConvoyMining. All datasets are available at the following sites.

- GeoLife:http://research.microsoft.com/en-us/projects/geolife/.
- Imis:http://chorochronos.datastories.org/.
- AIS:https://marinecadastre.gov/ais/.
- Chengdu:https://outreach.didichuxing.com/en/.

References

[1] Mohiuddin Ahmed, Raihan Seraj, and Syed Mohammed Shamsul Islam. 2020. The k-means algorithm: A comprehensive survey and performance evaluation. *Electronics* 9, 8 (2020), 1295.

[2] Stefan Atev, Grant Miller, and Nikolaos P Papanikolopoulos. 2010. Clustering of vehicle trajectories. *IEEE Transactions on Intelligent Transportation Systems* 11, 3 (2010), 647–657.

[3] Thapana Boonchoo, Xiang Ao, Yang Liu, Weizhong Zhao, Fuzhen Zhuang, and Qing He. 2019. Grid-based DBSCAN: Indexing and inference. *Pattern Recognition* 90 (2019), 271–284.

[4] Kevin Buchin, Anne Driemel, Natasja van de L'Isle, and André Nusser. 2019. klcluster: Center-based clustering of trajectories. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 496–499.

[5] Lu Chen, Yunjun Gao, Ziquan Fang, Xiaoye Miao, Christian S Jensen, and Chenjuan Guo. 2019. Real-time distributed co-movement pattern detection on streaming trajectories. *Proceedings of the VLDB Endowment* 12, 10 (2019), 1208–1220.

[6] Lisi Chen, Shuo Shang, Christian S Jensen, Bin Yao, and Panos Kalnis. 2020. Parallel semantic trajectory similarity join. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 997–1008.

[7] Zhenhua Chen, Yongjian Yang, Liping Huang, En Wang, and Dawei Li. 2018. Discovering urban traffic congestion propagation patterns with taxi trajectory data. *IEEE Access* 6 (2018), 69481–69491.

[8] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Vol. 96. 226–231.

[9] Qi Fan, Dongxiang Zhang, Huayu Wu, and Kian-Lee Tan. 2016. A general and parallel platform for mining co-movement patterns over large-scale trajectories. *Proceedings of the VLDB Endowment* 10, 4 (2016), 313–324.

[10] Markus Götz, Christian Bodenstein, and Morris Riedel. 2015. HPDBSCAN: highly parallel DBSCAN. In *Proceedings of the workshop on machine learning in high-performance computing environments*. 1–10.

[11] Joachim Gudmundsson and Marc Van Kreveld. 2006. Computing longest duration flocks in trajectory data. In *ACM International Symposium on Advances in Geographic Information Systems*. 35–42.

[12] Ade Gunawan and M de Berg. 2013. A faster algorithm for DBSCAN. *Master's thesis* (2013).

[13] Yunheng Han, Weiwei Sun, and Baihua Zheng. 2017. COMPRESS: A comprehensive framework of trajectory compression in road networks. *ACM Transactions on Database Systems* 42, 2 (2017), 1–49.

[14] Danlei Hu, Ziquan Fang, Hanxi Fang, Tianyi Li, Chunhui Shen, Lu Chen, and Yunjun Gao. 2022. Estimator: An Effective and Scalable Framework for Transportation Mode Classification over Trajectories. *IEEE Transactions on Intelligent Transportation Systems* (2022).

[15] Hoyoung Jeung, Qing Liu, Heng Tao Shen, and Xiaofang Zhou. 2008. A hybrid prediction model for moving objects. In *IEEE International Conference on Data Engineering*. 70–79.

[16] Hoyoung Jeung, Heng Tao Shen, and Xiaofang Zhou. 2008. Convoy queries in spatio-temporal databases. In *IEEE International Conference on Data Engineering*. 1457–1459.

[17] Hoyoung Jeung, Man Lung Yiu, Xiaofang Zhou, Christian S Jensen, and Heng Tao Shen. 2008. Discovery of convoys in trajectory databases. *Proceedings of the VLDB Endowment* 1, 1 (2008), 1068–1080.

[18] Tianyi Li, Lu Chen, Christian S Jensen, and Torben Bach Pedersen. 2021. TRACE: Real-time compression of streaming trajectories in road networks. *Proceedings of the VLDB Endowment* 14, 7 (2021), 1175–1187.

[19] Tianyi Li, Ruikai Huang, Lu Chen, Christian S Jensen, and Torben Bach Pedersen. 2020. Compression of uncertain trajectories in road networks. *Proceedings of the VLDB Endowment* 13, 7 (2020), 1050–1063.

[20] Xiaohui Li, Vaida Ceikute, Christian S Jensen, and Kian-Lee Tan. 2012. Effective online group discovery in trajectory databases. *IEEE Transactions on Knowledge and Data Engineering* 25, 12 (2012), 2752–2766.

[21] Yuxuan Li, James Bailey, and Lars Kulik. 2015. Efficient mining of platoon patterns in trajectory databases. *Data & Knowledge Engineering* 100 (2015), 167–187.

[22] Zhenhui Li, Bolin Ding, Jiawei Han, and Roland Kays. 2010. Swarm: Mining relaxed temporal moving object clusters. *Proceedings of the VLDB Endowment* 3, 1–2 (2010), 723–734.

[23] Zhenhui Li, Bolin Ding, Jiawei Han, Roland Kays, and Peter Nye. 2010. Mining periodic behaviors for moving objects. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1099–1108.

[24] Gang Liu, Zeting Wang, Amelie Chi Zhou, and Rui Mao. 2024. Adaptive key partitioning in distributed stream processing. *CCF Transactions on High Performance Computing* 6, 2 (2024), 164–178.

[25] Yuexin Ma, Xinge Zhu, Sibo Zhang, Ruigang Yang, Wenping Wang, and Dinesh Manocha. 2019. Trafficpredict: Trajectory prediction for heterogeneous traffic-agents. In *AAAI Conference on Artificial Intelligence*, Vol. 33. 6120–6127.

[26] Shaaban Mahran and Khaled Mahar. 2008. Using grid for accelerating density-based clustering. In *IEEE International Conference on Computer and Information Technology*. 35–40.

[27] Faisal Orakzai, Toon Calders, and Torben Bach Pedersen. 2019. k/2-hop: fast mining of convoy patterns with effective pruning. *Proceedings of the VLDB Endowment* 12, 9 (2019), 948–960.

[28] Faisal Orakzai, Torben Bach Pedersen, and Toon Calders. 2021. Distributed mining of convoys in large scale datasets. *GeoInformatica* 25, 2 (2021), 353–396.

[29] Shahbaz Rezaei and Xin Liu. 2019. Deep learning for encrypted traffic classification: An overview. *IEEE Communications Magazine* 57, 5 (2019), 76–81.

[30] Alex Rodriguez and Alessandro Laio. 2014. Clustering by fast search and find of density peaks. *science* 344, 6191 (2014), 1492–1496.

[31] Mehul A Shah, Joseph M Hellerstein, Sirish Chandrasekaran, and Michael J Franklin. 2003. Flux: An adaptive partitioning operator for continuous query systems. In *Proceedings 19th International Conference on Data Engineering*. IEEE, 25–36.

[32] Geza Szabo, Istvan Szabo, and Daniel Orincsay. 2007. Accurate traffic classification. In *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*. 1–8.

[33] Lu-An Tang, Yu Zheng, Jing Yuan, Jiawei Han, Alice Leung, Chih-Chieh Hung, and Wen-Chih Peng. 2012. On discovery of traveling companions from streaming trajectories. In *IEEE International Conference on Data Engineering*. 186–197.

[34] Andreas Tritsarolis, George-Stylianos Theodoropoulos, and Yannis Theodoridis. 2021. Online discovery of co-movement patterns in mobility data. *International Journal of Geographical Information Science* 35, 4 (2021), 819–845.

[35] Cheng-Fa Tsai, Chien-Tsung Wu, and Shengyong Chen. 2009. GF-DBSCAN; a new efficient and effective data clustering technique for large databases. In *WSEAS International Conference. Proceedings. Mathematics and Computers in Science and Engineering*. World Scientific and Engineering Academy and Society.

[36] Marcos R Vieira, Petko Bakalov, and Vassilis J Tsotras. 2009. On-line discovery of flock patterns in spatio-temporal data. In *ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 286–295.

[37] Chenxu Wang, Xin Yang, Tianyi Li, Jiaxing Wei, Pinghui Wang, Hongzhen Xiang, and Christain S Jensen. 2024. SWISP: Distributed Convoy Mining via Sliding Window-based Indexing and Sub-track Partitioning. In *International Conference on Data Engineering*. 4518–4531.

[38] Yiqiu Wang, Yan Gu, and Julian Shun. 2020. Theoretically-efficient and practical parallel DBSCAN. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2555–2571.

[39] Yida Wang, Ee-Peng Lim, and San-Yih Hwang. 2006. Efficient mining of group patterns from user movement data. *Data & Knowledge Engineering* 57, 3 (2006), 240–282.

[40] Hyunjin Yoon and Cyrus Shahabi. 2009. Accurate discovery of valid convoys from moving object trajectories. In *IEEE International Conference on Data Mining Workshops*. 636–643.

[41] Yanwei Yu, Lei Cao, Elke A Rundensteiner, and Qin Wang. 2014. Detecting moving object outliers in massive-scale trajectory streams. In *International Conference on Knowledge Discovery and Data Mining*. 422–431.

[42] Jie Zeng, Yong Xiong, Feiyang Liu, Junqing Ye, and Jinjun Tang. 2022. Uncovering the spatiotemporal patterns of traffic congestion from large-scale trajectory data: A complex network approach. *Physica A: Statistical Mechanics and its Applications* 604 (2022), 127871.

[43] Yu Zheng. 2015. Trajectory data mining: an overview. *ACM Transactions on Intelligent Systems and Technology* 6, 3 (2015), 1–41.