Colossal Trajectory Mining: A Unifying Approach to Mine Behavioral Mobility Patterns

Matteo Francia*, Enrico Gallinucci*, Matteo Golfarelli*

DISI – University of Bologna, Cesena, Italy

Abstract

Spatio-temporal mobility patterns are at the core of strategic applications such as urban planning and monitoring. Depending on the strength of spatio-temporal constraints, different mobility patterns can be defined. While existing approaches work well in the extraction of groups of objects sharing fine-grained paths, the huge volume of large-scale data asks for coarse-grained solutions. In this paper, we introduce Colossal Trajectory Mining (CTM) to efficiently extract heterogeneous mobility patterns out of a multidimensional space that, along with space and time dimensions, can consider additional trajectory features (e.g., means of transport or activity) to characterize behavioral mobility patterns. The algorithm is natively designed in a distributed fashion, and the experimental evaluation shows its scalability with respect to the involved features and the cardinality of the trajectory dataset.

Keywords: Trajectory mining, Mobility patterns, Frequent itemset mining

1. Introduction

The spreading of Internet of Things and mobile devices (Vitali et al., 2021) stimulated the rise of applications based on mobility patterns, such as urban mobility and traffic planning (Kumar et al., 2018), and moving objects (MOs) profiling and linking (Francia et al., 2020a,b) where objects moving in similar locations can share interests or relationships.

*Corresponding author.

Email addresses: m.francia@unibo.it (Matteo Francia), enrico.gallinucci@unibo.it (Enrico Gallinucci), matteo.golfarelli@unibo.it (Matteo Golfarelli)

Since different applications require identifying different mobility behaviors, researchers tailored a plethora of specific patterns. However, the need for a *unifying* analytic framework is well-understood and debated in the literature (Ding et al., 2018; Kwakye, 2020). Ad-hoc implementations require ad-hoc extract-transform-load workflows, processing, and analysis; each of these steps entails expertise to maintain and operate each algorithm. Conversely, a holistic and unifying analysis solution is an important step towards applications that involve massive trajectory data (Ding et al., 2018) also enriched from heterogeneous sources.

Generally speaking, a *mobility pattern* captures behaviors that frequently occur among MOs; each MO follows a trajectory that can be partially or completely shared with others. Mobility patterns can be classified by the strength of the constraints on spatial and temporal proximity. All patterns require spatial proximity to be satisfied for a sufficient length of the trajectories. Conversely, temporal proximity is not always mandatory: a mobility pattern can be interesting even if the same path has been traveled at different times. Table 1 reports four widely studied patterns covered by our approach and shows application examples. A co-location (Bao et al., 2022) is a group of objects located at the same points at any time (e.g., customers frequenting the same shops on different days). Similarly, a flow (Han et al., 2015) is a co-location group where the path is contiguous (e.g., individuals moving through adjacent road segments). A swarm (Li et al., 2010) is a group of objects moving within the same points at the same, possibly non-consecutive, time instants (e.g., individuals attending the same sport events each week). Similarly, a convoy (Jeung et al., 2008) is a swarm group sharing (at least some) consecutive timestamps (e.g., individuals sharing a means of transport).

Most of the approaches in the literature focus on a specific pattern and a few unifying approaches have been introduced. Phan et al. (2016) and Fan et al. (2016) target small groups of objects sharing fine-grained paths, however, the huge volume of large-scale data (e.g., hundreds of thousands of trajectories spanning the entire USA) asks for coarse-grained solutions. For instance, if our goal is to extract the groups of people flying across countries (e.g., USA has 50 states and 328 million inhabitants) or moving across city neighborhoods (e.g., Milan has 88 neighborhoods and 3 million inhabitants), meaningful groups are the ones in the order of hundreds/thousands of individuals. Indeed, extracting groups of 10 people given an average domestic flight of 150 people would return $\binom{150}{10} = 1.2 \cdot 10^{15}$ groups.

Table 1: Mobility patterns and examples of applications

Pattern	Application
Co-location (Bao et al., 2022)	Frequent POI/route
Flow (Han et al., 2015)	Historic traffic analysis
Swarm (Li et al., 2010)	Event & Gathering
Convoy (Jeung et al., 2008)	Online traffic analysis

Large-scale datasets demand a big-data solution, since (i) it could be impossible to load the entire dataset into main memory at once, and (ii) even if the dataset could fit into memory, millions of trajectories generate an exponential number of patterns. This is especially true for datasets where the number of trajectories, as well as their length in terms of points, is very large. Furthermore, Phan et al. (2016) and Fan et al. (2016) only address spatio-temporal patterns, preventing the possibility to characterize groups also in terms of additional features (e.g., means of transport, activity).

We introduce a novel Colossal Trajectory Mining (CTM) approach that generalizes many of the previous approaches while overcoming their limitations and integration effort. The main contributions of CTM are the following:

- it is a unifying framework for the extraction of heterogeneous mobility patterns;
- it is suitable for applications working on datasets with a huge number of (long) trajectories traveling across limited spatial regions (i.e., #trajectories $\gg$ #regions);
- it generalizes the mobility patterns reported in Table 1 through generic spatio-temporal constraints;
- it characterizes MOs through a tessellation including spatial and temporal features as well as additional features that enable the comprehension of semantic mobility behaviors (Yan et al., 2013) (e.g., characterizing mobility behaviors by means of transport or activity)[1];

[1]Note that the tessellation is given as input and the tuning of its granularity is out of the scope of the paper (see Section 2); we introduce an algorithm to mine co-movement patterns out of any (possibly semantic) tessellation.

- it does not require similarity computation between trajectories, preventing the design of burdensome metrics;
- it is implemented following the big-data paradigm, enabling high scalability in terms of both input trajectories and extracted patterns.

We emphasize that the terms "*colossal* (trajectory) mining" and "*big* data" have different meanings. Colossal (Pan et al., 2003) is an application of mining techniques to datasets having the number of dimensions (columns) sensibly higher than the number of instances (rows)[2]; colossal mining can require big data techniques, where big data refers to datasets (that produce results) that are too large to be dealt with by centralized data-processing applications.

The remainder of the paper is organized as follows. In Section 2, we position and compare our approach with respect to the related literature. In Section 3, we briefly describe CTM and motivate its adoption. Then, we formalize the theoretical foundations of our approach in Section 4, and we provide its implementation in Section 5. In Section 6, we assess the effectiveness and efficiency of CTM by leveraging both real-world and synthetic case studies. Finally, we summarize our approach and future research directions in Section 7.

2. Related work

CTM intersects frequent itemset mining and clustering.

Frequent Itemset Mining uncovers co-occurrences among the items in a transaction dataset (Agrawal and Srikant, 1994). FIM has been also applied to trajectory data to find routes (Qiu and Pi, 2016; Fu et al., 2017) or regions (Zheng et al., 2018) frequently trafficked by MOs. This is an *orthogonal* problem to extracting groups of objects moving together (e.g., a trajectory might support multiple routes and a route might be supported by dissimilar trajectories).

Colossal itemset mining is a branch of FIM that computes FI in highly-dimensional datasets; for instance, a dataset with 30 transactions/rows/instances each with 10^7 items/columns (Apiletti et al., 2017).

[2]This happens, for example, in biological datasets such as gene expression datasets that may contain 10^5 columns but only 10^3 rows (Pan et al., 2003).

Previous colossal itemset mining algorithms, such as Carpenter (Pan et al., 2003), generate frequent closed itemsets in a depth-first fashion. In a centralized architecture, Carpenter exploits an in-memory accumulator in which all patterns are stored; this cannot scale to data that do not fit into main memory: even if the transaction dataset can fit into main memory in a single machine, the exponential number of co-movement patterns might not. Apiletti et al. (2017) distribute the branches of the depth-first exploration over distributed executors. However, (i) this results in highly unbalanced partitions, (ii) a centralization mechanism must be iteratively applied to discard redundant branches (i.e., patterns are redundantly generated), and (iii) shuffling all the patterns is a data-intensive operation. While other algorithms for colossal itemset mining have been recently proposed (Pan et al., 2003; Zaki and Zulkurnain, 2018; Vanahalli and Patil, 2019), *none* address the extraction of constrained co-movement patterns.

Clustering groups trajectories such that intra-group similarity is maximized and inter-group similarity is minimized. Clustering techniques are characterized by the *rationale* and the *similarity metric*.

As to the rationale, *flow-based* clustering can group trajectories with very limited overlapping (Han et al., 2012, 2015); for instance, after a map-matching phase (Francia et al., 2019), cars traveling different segments of the same road determine a flow along the road. *Partitioning* clustering groups whole trajectories (Han et al., 2017; Kumar et al., 2018; Wang et al., 2019) and does not preserve inter-cluster patterns (e.g., common sub-trajectories). Indeed, while a trajectory is grouped with similar ones, still it can share interesting paths with trajectories from other clusters. *Overlapping* clustering (Lee et al., 2007; Tampakis et al., 2019; Ailin et al., 2019) splits trajectories into sub-trajectories and groups them. However, even if at a smaller scale, this leads to the loss of so-called rare patterns, namely meaningful spatial events occurring with a low frequency (Koh and Ravana, 2016). As a consequence, all these rationales are limited since they do not preserve groups sharing a few points (which can be strategic).

As to the similarity definition, the higher the expressiveness, the higher the computational complexity. The Euclidean distance (Van de Geer, 1995) has linear complexity in trajectory length but requires equally-long trajectories and does not detect time shifts. DTW (Rakthanmanon et al., 2012) overcomes these limitations and is robust to missing points but at the cost of quadratic complexity. Further distance func-

tions are Hausdorff (Sim et al., 1999) (the greatest of all the distances from a point in one trajectory to the closest point in the other trajectory), LCSS (Vlachos et al. (2002); similarity is expressed in terms of the longest common subsequence of two trajectories), Frechet (Agarwal et al. (2018); the smallest of the maximum pairwise distances), and road based (da Silva et al. (2020); the minimum number of network paths between two trajectories in a certain time window). Different distance functions produce different clusterings; Vlachos et al. (2002) compare LCSS, Euclidean, and DTW. However, these *spatio-temporal* distances cannot be directly adopted in our approach for the following reasons:

- they do not consider further geometric (e.g., speed or direction) or semantic (e.g., point types or means of transport) features;
- they cannot be applied at different aggregation levels (e.g., neighborhoods and cities);
- they cannot be applied to all co-movement patterns (e.g., LCSS, which returns as similar trajectories sharing a contiguous path, is not suitable for the co-location pattern where path adjacency is not required).

Since "classic" clustering algorithms *do not* enforce a definition of co-movement patterns (e.g., cardinality, time span, cohesion, spatial and/or temporal contiguity), flock (Gudmundsson and van Kreveld, 2006), convoy (Jeung et al., 2008), and swarm (Aung and Tan, 2010) patterns have been formalized. Since swarm (Aung and Tan, 2010) groups objects moving in spatial proximity for a given amount of possibly-non-contiguous time, classical clustering algorithms can be referred to as swarms if no minimal duration is enforced.

Distinguishing features of CTM. Since MOs outnumber the regions of interest, CTM is a novel perspective in the analysis of mobility patterns. The research on co-movement patterns is active; recent relevant papers work toward improving the mining efficiency through spatial hierarchical constraints (Helmi and Banaei-Kashani, 2020) or by providing pattern-specific strategies; for instance filtering convoys in either a centralized (Aung and Tan, 2010; Orakzai et al., 2019; Liu et al., 2021) or distributed (Orakzai et al., 2021; Tritsarolis et al., 2022) fashion. Our goal is providing a unifying framework for pattern analysis. The need for a unifying framework is well-understood and debated in

literature (Ding et al., 2018; Kwakye, 2020). Phan et al. (2016) and Fan et al. (2016) are close contributions to CTM and both extract spatio-temporal mobility patterns by discretizing time in bins, clustering MOs in every bin, and merging clusters across bins. Table 2 summarizes the comparison; the strengths of CTM are the following.

1. Transparently and homogeneously supporting semantic features to characterize behavioral mobility patterns (e.g., neighborhood and municipalities, means of transport, holiday or work day; Figures 1a and 1b). Geometric-only (e.g., spatio-temporal) tessellations are supported but they are only one of the possible tessellation types. This makes CTM extensible in the characterization of behavioral mobility patterns.

2. Extracting spatial, spatio-temporal, and semantic patterns — whereas Phan et al. (2016); Fan et al. (2016) require an *absolute* temporal dimension— by applying constraint and pruning strategies that allow to return several pattern types as well as to reduce the algorithmic search space.

3. Providing a native distributed solution to the problem that can scale up to real-world datasets and a lossless compression of the output patterns by leveraging frequent closed itemsets.

While CTM could be associated with grid-based clustering, automatically finding the best grid for a specific dataset/problem/co-movement pattern is out of the scope of the paper. To ease the computation of similarity, grid-based clustering uses a grid to build the clusters out of adjacent densely-populated cells. Setting the resolution grid is difficult (Parsons et al., 2004; Jin et al., 2022), especially in multi-dimensional datasets with heterogeneous densities across dimensions. In this case, even adaptive grids (Lu et al., 2005) are not a solution since they raise the following dataset- and problem-specific issues.

- *Dataset-specific*: computing the "best" tessellation requires to define what are the "best" co-movement patterns. Internal metrics measure how distinguishable clusters are without the need for external knowledge. For instance, the silhouette index measures whether crisp partitioning clusters are compact and well-separated (Zhu et al., 2010). However, co-movement patterns can

be overlapping since trajectories can contribute to multiple groups of MOs. External metrics require labels to find the "best" co-movement patterns out of the given dataset/tessellation (e.g., the "purity" of clusters through entropy); this type of supervised evaluation is usually leveraged to compare different grid-based algorithms (Parsons et al., 2004). Besides the metrics, defining the "best" co-movement patterns also depends on the dataset heterogeneity; e.g., taxis in (Yuan et al., 2010) produce homogeneous and precise trajectories, while Movebank (Kranstauber et al., 2011) collects data from thousands of studies.

- *Problem-specific*: co-movement patterns depend on (i) the goal of the analysis (e.g., monitoring co-movement patterns at single user levels at the scale of meters, or aggregated network data at the scale of kilometers); and (ii) the type of moving objects (e.g., to retrieve a convoy of 5 people moving in the same car a cell of 6 square meters could be enough, but this is not true to detect convoys of cars or trucks spanning hundreds of meters —for instance in a highway).

3. Approach overview

Modeling mobility pattern mining as a *colossal* frequent itemset problem is a novel research direction. The goal of Frequent Itemset Mining (Agrawal and Srikant, 1994), historically used for market-basket analysis, is to extract *items* (e.g., products) that often appear together in a dataset of *transactions* (e.g., receipts of product sales). Similar to the sales problem, FIM retrieves sets of trajectories (i.e., items) moving together in the same places (i.e., transactions). Colossal refers to the scenario in which the number of traveled places is sensibly lower than the number of trajectories (Pan et al., 2003).

Our approach is based on the following steps (Figure 2).

3.1. Abstracting trajectories

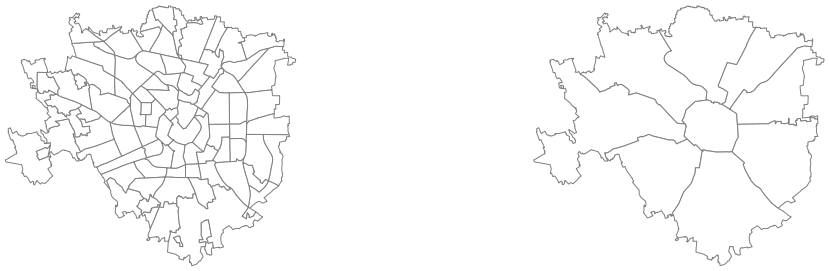
Raw trajectories are initially mapped to a multidimensional tessellation, a composition of multidimensional tiles (from now on, simply *tiles*) of *any* shape — with no overlaps and no gaps — that cover a multidimensional space; the tessellation *is not* necessarily a regular grid. Each dimension of the tessellation corresponds to a feature describing the raw trajectory points:

Table 2: Comparing co-movement patterns approaches by supported features and pattern types and distributed implementation

Features	Co-location	Flow	Convoy	Swarm	Distributed	Contributions
Any	✓	✓	✓	✓	✓	CTM
Space	-	✓	-	-	-	da Silva et al. (2020), Yang et al. (2022), Tritsarolis et al. (2021)
Space	✓	-	-	-	-	Lv et al. (2019), Tran et al. (2021)
Space	✓	-	-	-	✓	Fonseca-Galindo et al. (2022)
Space, Time	-	-	-	✓	-	Lee et al. (2007), Li et al. (2010), Han et al. (2017), Ailin et al. (2019), Li et al. (2023)
Space, Time	-	-	-	✓	✓	Hu et al. (2015), Tampakis et al. (2019)
Space, Time	-	-	✓	-	-	Aung and Tan (2010), Orakzai et al. (2019), Liu et al. (2021)
Space, Time	-	-	✓	-	✓	Orakzai et al. (2021), Tritsarolis et al. (2022)
Space, Time	-	-	✓	✓	-	Li et al. (2015), Phan et al. (2016), Helmi and Banaei-Kashani (2020)
Space, Time	-	-	✓	✓	✓	Fan et al. (2016)
Space, Time	-	✓	-	-	-	Han et al. (2015), Kumar et al. (2018), Wang et al. (2019)

common features are space and time, but additional features can be added to specialize each trajectory point (e.g., whether an individual is moving by car or by bicycle). This allows grouping trajectories on semantic and behavioral concepts (e.g., to capture groups of individuals moving across city neighborhoods with different means of transport). Mapping raw trajectories into the tessellation brings the following benefits.

- The tessellation granularity defines the level of the analysis; for instance, CTM transparently allows the extraction of patterns at neighborhood/city/country scales (Figure 1).
- The tessellation automatically compresses trajectories since a trajectory *moves through* a tile if at least one of its points belongs to the tile (i.e., there is no need to store consecutive points from the same tile more than once). This makes the computational complexity more related to the scale of tessellation rather than to the trajectories length. For this reason, CTM is particularly suited for applications working on a huge number of, potentially long, trajectories and a coarse tessellation.



(a) Neighborhoods (b) Municipalities

Figure 1: Administrative spatial tessellations of Milan.

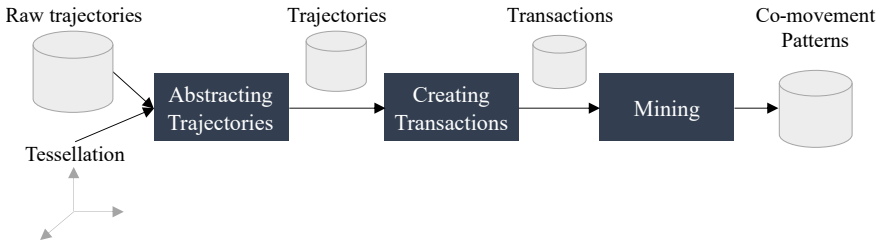


Figure 2: Overview of CTM.

- The tessellation implicitly defines similarity: the more tiles two trajectories share, the higher their similarity.

3.2. Creating transactions

To each tile corresponds a transaction including the trajectories (i.e., items) moving through it.

Example 1 (Trajectory and transaction). *With reference to Figure 3, where a regular tessellation is used for simplicity,* T_b, T_g, *and* T_r *are trajectories while* Q_{A1} *and* Q_{B2} *are transactions that correspond to tiles A1 and B2.*

$$T_b = (A1, B1, C1, D1, C1), T_g = (A1, B1, B2, B3, B4, A4),$$
$$T_r = (A1, B1, B2, B3, D3), Q_{A1} = \{T_b, T_g, T_r\}, Q_{B2} = \{T_g, T_r\}$$

Mapping T_g *into the tessellation allows its compression since consecutive points in tiles* $B2$ *and* $B3$ *are only stored once. The tessellation defines the granularity and the semantics of the analysis. If — for instance — the*

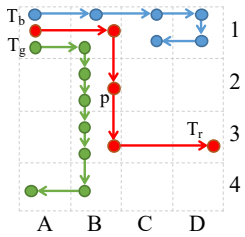


Figure 3: Trajectories moving through a tessellation.

user is also interested in distinguishing malls or restaurants or means of transport, space and time features alone are not enough. Additional features (e.g., placeType or transport) must be added to the tessellation. This is the strength of our approach: to consider custom features transparently. Indeed, tiles enable an extensible and transparent management of space, time, and additional features. For instance, in a 2D setting (where only space — latitude and longitude — is considered), tiles $A1$ *and* $A2$ *could represent the following bins/partitions*

$$A1 = (lat = 44.123, lon = 12.123)$$
$$B2 = (lat = 44.124, lon = 12.124)$$

while in a 4D setting (space, time, and means of transport) $A1, A2$ *could be*

$$A1 = (lat = 44.123, lon = 12.123, t = 2022/08/04\ 10:55:00, transport = walk)$$
$$B2 = (lat = 44.124, lon = 12.124, t = 2022/08/04\ 10:56:00, transport = bus)$$

While this modeling naturally allows the extraction of groups of trajectories moving in defined spatial regions, some might find more natural to do a "transposed" modeling using tiles as items and trajectories as transactions. In this case, the output would be the groups of places visited together rather than the group of trajectories traveling together.

Modeling trajectories as items and tiles as transactions makes our frequent itemset approach a *colossal* one since the number of tiles (i.e., transactions) is typically in the order of magnitudes smaller than the number of trajectories (i.e., items). This is a fair assumption to make: for instance, Milan has 88 neighborhoods with over $3 \cdot 10^6$ inhabitants (i.e., potential MOs). Obviously, our assumption is no more true when a very fine tessellation is adopted. For example, the Milan metropolitan area spans about $1500km^2$, corresponding to $1.5 \cdot 10^5$ uniform tiles with side $100m$ and $1.5 \cdot 10^7$ uniform tiles with side $10m$.

3.3. Mining co-movement patterns

Once a transaction dataset is created, the colossal itemset mining algorithm retrieves the sets of trajectories satisfying minimum cardinality (i.e., the number of trajectories), minimum support (i.e., the length of the shared path), and spatio-temporal constraints. Different mobility patterns can be obtained by specializing such constraints (as later shown).

Note that approaching this problem as a colossal itemset mining rather than a typical clustering one (see Section 2):

- avoids the computation of similarities that would make computational complexity explode for large datasets;
- exploits monotonicity properties (e.g., as the generation process proceeds, the cardinality of trajectory groups decreases while the length of the path shared by trajectories in the same group increases) to filter out invalid co-movement patterns without generating them (filtering strategies are detailed in Section 5.2).

4. Modeling co-movement patterns

We consider a dataset of raw trajectories, where each trajectory point is labeled with a set of features. Mandatory features are those needed to define the spatial or spatio-temporal location (e.g., latitude, longitude, and timestamp), but many other physical or semantic features might be available (e.g., speed, day of the week, and neighborhood).

Definition 1 (Raw trajectory and feature). *A raw trajectory P is a sequence of points $(p_1, \ldots, p_{|P|})$ generated by a MO. The feature space is a set of features $F = \{f_1, \ldots, f_{|F|}\}$ that is collected for each point; it represents a multidimensional space where each dimension corresponds to a feature.*

Definition 2 (Tessellation). *Given a dataset of raw trajectories associated with the features F, we call tessellation a multidimensional partitioning $S = \{s_1, \ldots, s_m\}$ of the feature space F. Each tile of the tessellation is identified by an id and is characterized by an interval/set of values for each continuous/nominal feature in F. For each feature f, the function $dist_f(s_i, s_j)$ computes the distance between two tiles on the tessellation. Two tiles s_i, $s_j \in S$ are adjacent ($s_i \cong_f s_j$) if $\forall s_z \in S$ it is $dist_f(s_i, s_j) \leqslant dist_f(s_i, s_z) + dist_f(s_z, s_j)$.*

Note that $dist_f()$ is a distance between tiles used to characterize different mobility patterns. Since in a colossal problem the number of tiles is order of magnitude lower than the number of trajectories and trajectory points, computing distances between tiles entails a computational complexity that is by far lower than working on raw trajectories. In our implementation, the distance function is

$$dist_f(s_i, s_j) = \begin{cases} f \text{ is ordinal} & geod(s_i, s_j) \\ f \text{ is nominal} \wedge s_i = s_j & 0 \\ f \text{ is nominal} \wedge s_i \neq s_j & \infty \end{cases} \quad (1)$$

where $geod(s_i, s_j)$ is the geodesic distance (Bouttier et al., 2003) computed on the tessellation, which is the number of tiles along the shortest path of neighboring tiles connecting s_i and s_j. Although in principle any distance metric can be leveraged (e.g., Haversine or Euclidean), using the geodesic distance simplifies the formalization (and implementation) of co-movement patterns. As to nominal features (e.g., monitoring urban traffic on weather conditions such as rainy/sunny) that do not allow ordering, the distance function allows the selection of homogeneous tiles, and in turn, searches for co-movement patterns characterized by the same nominal values.

Definition 3 (Trajectory). *Given a raw trajectory P and a tessellation S, we define the trajectory T corresponding to P in S as the sequence of tiles $(s_1, \ldots, s_{|T|})$ such that a tile s is added to T if at least a point $p \in P$ is in s. A point p is in the tile s if, for each feature in F, the values of the feature for s contain the corresponding feature value characterizing p.*

A trajectory is an abstraction of a raw trajectory at the grain defined by the tessellation. This determines a first data compression: a single tile instance is added to T if consecutive points fall in that tile, thus $|P| \geqslant |T|$.

Definition 4 (Tile connection). *Two tiles s_i, s_j in the tessellation S are connected ($s_i \leftrightarrow_f s_j$) if there exists a path of adjacent tiles in S connecting them in f.*

Example 2 (Tessellation, tile, and trajectory). *With reference to Figure 3, the grid is a tessellation involving the spatial feature f^{sp} which characterizes 16 tiles (i.e., cells from $A1$ to $D4$); $dist_{f^{sp}}(A1, A2) = 1$ and $dist_{f^{sp}}(A1, A3) = 2$; the point p is in the tile $B2$ and the trajectory $T_r = (A1, B1, B2, B3, D3)$ corresponds to the red raw trajectory within the tessellation.*

To enable the formalization of CTM as a colossal itemset mining problem, we need to introduce *transactions* and *items*. Each transaction corresponds to a tile and includes the items/trajectories moving through that tile.

Definition 5 (Item, itemset, and transaction). *Given a trajectory dataset $\mathcal{T}$ and a tessellation S, each trajectory represents an item, and a set I of trajectories is an itemset. We define the transaction for a tile $s \in S$ as the itemset containing all the trajectories having at least a point in s. $\mathcal{Q}$, the set of all transactions, is a transaction dataset.*

The maximum number of transactions depends on the number of tiles (i.e., $|\mathcal{Q}| = |S|$). Furthermore, as transactions are *sets* of items, the data is further compressed: if a tile is traversed more than once by a trajectory, the transaction corresponding to the tile will contain the trajectory/item only once.

As for FIM, our goal is identifying the itemsets contained in a large number of transactions. This property is captured by the support function.

Definition 6 (Support). *Given a transaction dataset $\mathcal{Q}$, the support $sup(I) \subseteq \mathcal{Q}$ of an itemset I is the set of transactions containing I.*

Example 3 (Itemset, transaction, and support). *With reference to Figure 3, $Q_{A1} = \{T_b, T_g, T_r\}$ is the transaction corresponding to the tile $A1$, $I = \{T_g, T_r\}$ is an itemset, and $sup(I) = \{Q_{A1}, Q_{B1}, Q_{B2}, Q_{B3}\}$.*

Definition 7 (Frequent and closed itemsets). *An itemset is frequent (FI) if $|sup(I)| \geqslant mSup$, where $mSup$ is the minimum number of transactions to consider the itemset as frequent. A frequent itemset is closed (FCI) if there exists no superset with the same support.*

FCIs provide a lossless compression of FIs (Pei et al., 2000) (i.e., the output is non-redundant and the complete set of FIs is recoverable) which are exponential in the number of trajectories/items (Agrawal and Srikant, 1994). Working with FCIs rather than FIs simplifies data analysis (Francia et al., 2020b).

Definition 8 (Co-movement pattern). *A co-movement pattern I is an FCI such that $|I| \geqslant mCrd$, where $mCrd$ is the minimum number of trajectories to consider a FCI as a co-movement pattern.*

Table 3: Characterization of co-movement patterns in terms of spatial (f^{sp}) and temporal (f^{tm}) features and shape constraints

Pattern	Feature	Shape constraints
Co-loc.	f^{sp}	None
Flow	f^{sp}	$\exists S' \subseteq sup(I)$ s.t. $\forall s_i, s_j \in S', s_i \leftrightarrow_{f^{sp}} s_j \wedge \lvert S' \rvert \geqslant mSup$
Swarm	f^{sp}, f^{tm}	None
Convoy	f^{sp}, f^{tm}	$\exists S' \subseteq sup(I)$ s.t. $\forall s_i, s_j \in S', s_i \leftrightarrow_{f^{tm}} s_j \wedge \lvert S' \rvert \geqslant mSup$

By definition, a co-movement pattern has at least $mCrd$ trajectories and is at least $mSup$ tiles long. This "basic" co-movement pattern can be specialized into those reported in Table 1 depending on the involved features, either space (f^{sp}) or space and time (f^{tm}), and on the additional shape constraints described in Table 3. Figure 4 depicts an example of each pattern.

(a) Co-location (Francia et al., 2020b): trajectories sharing spatial points (e.g., individuals working and shopping in the same places, even at different times). No shape constraint is needed and the tessellation must include the space feature.

(b) Flow (Han et al., 2015): trajectories sharing *contiguous* spatial points (e.g., traffic over contiguous roads). The shape constraint ensures tile connection on the space dimension.

(c) Swarm (Li et al., 2010): trajectories sharing spatial points at the same time (e.g., individuals being occasionally together). No shape constraint is needed and the tessellation must include space and time features.

(d) Convoy (Jeung et al., 2008): trajectories sharing spatial points *continuously* in time (e.g., individuals moving together[3]). The shape constraint ensures tile connection on the time dimension.

As summarized in Table 3, co-location and flow are defined over a spatial feature (f^{sp}; i.e., they are required to happen in the same spatial tile), while swarm and convoy require both spatial (f^{sp}) and temporal (f^{tm}) features

[3]While the original formalization involves individuals *always* moving together, this constraint has been relaxed to a *sufficient* amount of contiguous temporal tiles (Fan et al., 2016). Otherwise, meaningful patterns can be lost due to noisy trajectories (even a single missing trajectory point).

(i.e., they are required to happen in the same spatio-temporal tiles). In other words, while for swarm and convoy it is necessary to be in the same spatio-temporal tile, co-location and flow only require to be in the same spatial tile (e.g., to be in the same location even if at different times).

A strong point of CTM is the capability to homogeneously manage space, time, and any additional semantic feature. Although the simplest representation of the space and time features is a regular binning of their absolute values, CTM allows adopting abstractions richer in semantics as long as these determine a tessellation (i.e., a partitioning) of space and time.

4.1. Handling additional features through behavioral constraints

Co-movement patterns are described by more than space and time features alone. Nominal, ordinal, and continuous features can further characterize behaviors that trajectories must share. Individuals could share means of transport and activities, or move and stand together; for instance, groups of people can take the same means of transport to reach the city center and go shopping in the same malls. To create a co-movement pattern, feature values must be the same in the path shared by the MOs (e.g., to be in the same group, individuals should share the same means of transport).

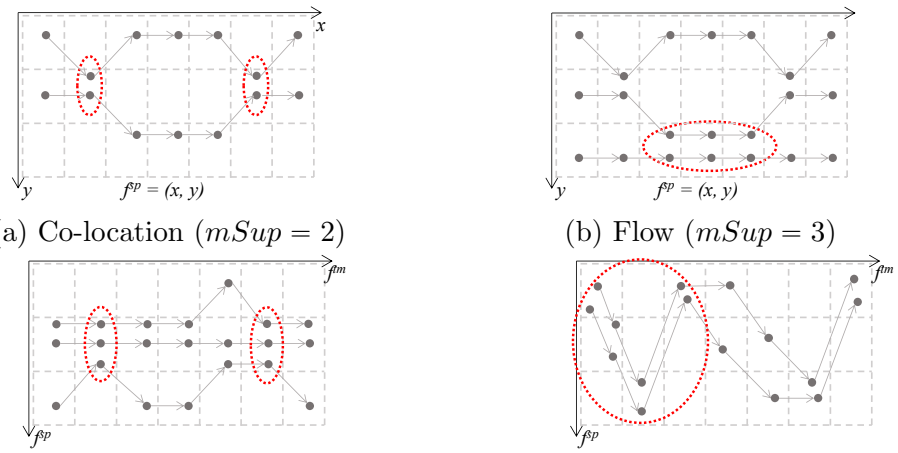
Characterizing co-movement patterns with additional features means imposing additional constraints, which we call *behavioral* constraints. Note that behavioral constraints are more expressive than filtering trajectories based on a specific feature value since they further characterize objects that *behaves similarly* while moving in space and time; for instance, behavioral features are highly important in the linkage/anonymization of mobility data (Jin et al., 2023). In CTM, behavioral constraints are transparently enforced by simply extending the input tessellation with additional features (see Definition 3): the tile directly models the features in the tessellation. More formally, two or more trajectories *share* a tile s if they have at least a point in s. Then, the support of an itemset I (i.e., a set of trajectories), includes all and only the transactions (i.e., the tiles) shared by the trajectories.

The behavioral constraints must be computed jointly with the spatio-temporal ones, and not before/after running CTM.

- A *pre-processing strategy* could apply behavioral constraints directly to the raw trajectory dataset (i.e., before applying CTM). At this stage, only filters on single trajectories can be applied. However, behavioral constraints require the portion of space/time shared by groups of trajectories to be known.

- A *post-processing strategy* could compute co-movement patterns considering spatio-temporal features first and then discarding the returned co-movement patterns that do not fulfill behavioral constraints. However, co-movement patterns are closed by definition. If a co-movement pattern becomes unfeasible due to behavioral constraints some of its non-closed subsets will become closed and must be returned. Since only closed itemsets are computed and returned, closeness and feasibility should be tested for all the subsets; however, the cardinality of the subsets is exponential in the pattern support. Alternatively, an approach returning non-closed itemsets too should be adopted, but this would imply much heavier enumeration (Lucchese et al., 2006).

Example 4. *Suppose we are looking for co-location patterns (i.e., individuals moving through some places without time constraints) characterized by stop-move phases, where stop points are used to infer the purpose of a trip and moves can provide information such as direction and mode of transport (Wang and McArthur, 2018). Let* $mSup = 3$ *and consider three individuals Alice, Charles, and Paul sharing the same route of 3 tiles* $A1, B1$*, and* $C1$*. On the one hand, Alice and Charles stop by stores in tiles* $A1$ *and* $C1$ *and walk by* $B1$*. On the other hand, Paul is only walking through the three tiles. Thus, considering the speed feature, only* $\{Alice, Charles\}$ *should emerge as a co-movement pattern, while* $\{Alice, Charles, Paul\}$ *(which is valid only from the spatio-temporal point of view) should be discarded. A pre-processing technique could be to discretize speed values and run an instance of CTM for each speed bin value; however, the pattern related to* $\{Alice, Charles\}$ *would be lost since it involves both walking and stops; more formally,* $sup(\{Alice, Charles\}) = |\{Q_{B1}\}| < mSup$ *when walking and* $sup(\{Alice, Charles\}) = |\{Q_{A1}, Q_{C1}\}| < mSup$ *when shopping. Alternatively, a post-processing technique could be to run CTM with only spatio-temporal features, return both* $\{Alice, Charles, Paul\}$ *and* $\{Alice, Charles\}$*, and finally discard the former by analyzing the speed feature. However, since* $sup(\{Alice, Charles, Paul\}) = sup(\{Alice, Charles\})$*, the pattern* $\{Alice, Charles\}$ *is not closed (see Definition 7) and cannot be returned; as result, the valid co-movement pattern* $\{Alice, Charles\}$ *is lost.*



(a) Co-location ($mSup = 2$)

(b) Flow ($mSup = 3$)

(c) Swarm($mCrd = 3, mSup = 2$)

(d) Convoy ($mSup = 4$)

Figure 4: Co-movement patterns (unless specified, $mCrd = 2$).

5. Mining co-movement patterns

We initially remind that trajectories are mapped to items and tiles are mapped to transactions. For the sake of clarity, henceforth we only refer to the items and transactions.

When the number of items is much larger than the number of transactions, searching for frequent itemsets by enumerating all itemsets with an *Apriori-like strategy* (Agrawal and Srikant, 1994) can be unfeasible or very inefficient. In this case, it is convenient to adopt a *row-enumeration-like strategy* (Pan et al., 2003), which is enumerating all the possible transaction sets. Each set of transactions $\bar{\mathcal{Q}}$ corresponds to the intersection of the itemsets appearing in those transactions.

Definition 9 ($\bar{\mathcal{Q}}$-Itemset). *Given a transaction dataset $\mathcal{Q}$, a $\bar{\mathcal{Q}}$-itemset is the largest itemset supported by all transactions $\bar{\mathcal{Q}} \subseteq \mathcal{Q}$.*

Each transaction corresponds to a $\bar{\mathcal{Q}}$-itemset with $|\bar{\mathcal{Q}}| = 1$; a generic $\bar{\mathcal{Q}}$-itemset is the intersection of the itemsets in $\bar{\mathcal{Q}}$.

Example 5 ($\bar{\mathcal{Q}}$-Itemset). *With reference to Figure 3, $I = \{T_g, T_r\}$ is a $\bar{\mathcal{Q}}$-itemset with $\bar{\mathcal{Q}} = \{Q_{A1}, Q_{B1}, Q_{B2}, Q_{B3}\}$.*

Theorem 1 (Closeness of $\bar{\mathcal{Q}}$-itemset). *$\bar{\mathcal{Q}}$-itemsets are all and only the closed itemsets.*

See Proof 1 in Appendix A.

CTM extracts co-movement patterns through a breadth-first enumeration of $\bar{\mathcal{Q}}$-itemsets: starting from single transactions, CTM progressively intersects them with further transactions. $\bar{\mathcal{Q}}$-itemsets associated with a larger set of transactions are characterized by lower cardinalities and larger supports (intuitively, fewer trajectories sharing more tiles). The enumeration process can be represented as an enumeration tree, where each node N corresponds to a distinct set of transactions $\bar{\mathcal{Q}}$. We call *enumeration sequence* the path leading from the root to a node N.

Naively enumerating the whole tree is inefficient, we exploit several pruning mechanisms to limit the enumerated portion of the tree. We conceive CTM as a parallel and big-data approach for co-movement pattern mining, thus it exploits solutions that would be neither necessary nor optimal in a centralized implementation. Specifically, CTM:

- adopts a breadth-first enumeration approach to fully exploit task parallelization and workload balancing;
- adopts local pruning criteria only to avoid centralized checks that would limit parallelization;
- adopts spatio-temporal pruning criteria that have been specifically devised for trajectories-related patterns;
- broadcasts the transaction dataset $\mathcal{Q}$ to locally compute the itemset support. This is a reasonable assumption since following our approach, its size is limited and can fit the central memory (see Section 6).

We now introduce the enumeration process, the pruning techniques, and, finally, how we distribute the algorithm to support scalable generation of co-movement patterns.

5.1. The enumeration process

Algorithm 1 introduces the pseudo-code of CTM which takes as inputs the transaction dataset $\mathcal{Q}$, the minimum cardinality $mCrd$, and the minimum support $mSup$. For the sake of simplicity, we assume $\mathcal{P}$ (Line 1) to be a *global* variable used as an accumulator of nodes corresponding to the co-movement patterns to be returned. $\mathcal{L}$ (Line 2) is a first-in-first-out (FIFO) queue that ensures breadth-first enumeration. We use the "dot" notation to indicate the

Algorithm 1 CTM

Require: $\mathcal{Q}$: transactions, $mSup$: minimum support, $mCrd$: minimum cardinality
Ensure: $\mathcal{P}$: co-movement patterns

1: $\mathcal{P} \leftarrow \varnothing$ ▷ Global accumulator of co-movement patterns
2: $\mathcal{L} \leftarrow \varnothing$ ▷ FIFO queue of $\bar{\mathcal{Q}}$-itemsets
3: **for each** $Q \in \mathcal{Q}$ **do** ▷ For each transaction
4: $N \leftarrow newNode()$ ▷ create an enumeration node
5: $N.I \leftarrow Q$ ▷ initialize the node $\bar{\mathcal{Q}}$-itemset with a transaction
6: $N.CT \leftarrow \{Q\}$ ▷ set the Covered Transactions
7: $N.RT \leftarrow \{Q' | Q' \in \mathcal{Q}, id(Q') > id(Q)\}$ ▷ set the Remaining Transactions
8: $\mathcal{L}.enqueue(N)$ ▷ enqueue the node for extension
9: **while** $|\mathcal{L}| > 0$ **do** ▷ While an extendable node exists
10: $N \leftarrow \mathcal{L}.dequeue$ ▷ remove it from the queue
11: $\bar{\mathcal{L}} \leftarrow Extend(N, mSup, mCrd)$ ▷ extend it
12: $\mathcal{L}.enqueue(\bar{\mathcal{L}})$ ▷ add the new nodes to the queue
13: **return** $\mathcal{P}$ ▷ Return the co-movement patterns

information in each node N. Each enumeration node N is associated with a $\bar{\mathcal{Q}}$-itemset $N.I$. Such $\bar{\mathcal{Q}}$-itemset is obtained by intersecting the transactions building up the enumeration sequence $N.CT$ (*Covered Transactions*), which is $N.CT = \bar{\mathcal{Q}}$. Let $N.RT$ (*Remaining Transactions*) be the transactions whose enumeration is expected in the N subtree. Note that enumeration nodes are unique: they can lead to the same itemset $N.I$ but they will have different $N.CT$ and $N.RT$. Nodes that potentially contain co-movement patterns are added to the queue (Line 8). While at least an extendable node exists (Line 9), a node is removed from the queue (Line 10) and extended (Line 11); the resulting nodes are appended in $\mathcal{L}$ (Line 12). Finally, the nodes associated with co-movement patterns are returned (Line 13).

Algorithm 2, applied in Algorithm 1 Line 11, shows how enumeration nodes are extended. First of all, if the node corresponds to a valid co-movement pattern (Line 1), it is added to the global accumulator (Line 2). Then, an empty queue is initialized to store the extendable nodes (Line 3), which are enumerated by N if the latter can potentially generate co-movement patterns (Line 4). The $Check()$ function, which verifies validity and extensibility of a pattern, is discussed in Section 5.2. Before enumerating new nodes, Lines 5-7 avoid useless extension steps: transactions Y shared by all the items in the $\bar{\mathcal{Q}}$-itemset (Line 5) will generate no new $\bar{\mathcal{Q}}$-itemsets since intersecting $N.I$ with the transactions in Y would produce $N.I$ itself by definition. Thus, transactions in Y should not be intersected; they can be removed from $N.RT$ and directly added into $N.CT$. For each of the transactions that have yet to be covered (Line 9), a new node is generated by intersecting the items in $N.I$ with the ones in the transaction (Line 11).

Algorithm 2 Extend

Require: N: enumeration node, $mSup$: support, $mCrd$: cardinality
Ensure: $\mathcal{L}$: Extended $\bar{\mathcal{Q}}$-itemsets
1: **if** $Check(N, \text{“val”}, mSup, mCrd) = true$ **then**
▷ If N refers to a co-movement pattern
2: $\mathcal{P} \leftarrow \mathcal{P} \cup \{N\}$ ▷ store it
3: $\mathcal{L} \leftarrow \varnothing$ ▷ Initialize a node queue
4: **if** $Check(N, \text{“ext”}, mSup, mCrd) = true$ **then**
▷ If N's subtree potentially contains co-movement patterns
5: $Y \leftarrow sup(N.I) \cap N.RT$ ▷ Transactions shared by all items in $N.I$
6: $N.CT \leftarrow N.CT \cup Y$ ▷ are added to the covered set
7: $N.RT \leftarrow N.RT \backslash Y$ ▷ and removed from those to be extended
8: $RT_{next} \leftarrow N.RT$ ▷ Initialize RT_{next}
9: **for each** $Q \in N.RT$ **do** ▷ For each remaining transaction
10: $N' \leftarrow newNode()$ ▷ create an enumeration node N'
11: $N'.I \leftarrow N.I \cap Q$ ▷ set the $\bar{\mathcal{Q}}$-itemset
12: $N'.CT \leftarrow N.CT \cup \{Q\}$ ▷ extend the Covered Transactions
13: $RT_{next} \leftarrow RT_{next} \backslash \{Q\}$ ▷ reduce the Remaining Transactions
14: $N'.RT \leftarrow RT_{next}$ ▷ and assign them to N'
15: $\mathcal{L}.enqueue(N')$ ▷ enqueue the node for extension
16: **return** $\mathcal{L}$ ▷ Return the extended nodes

Algorithm 3 Check

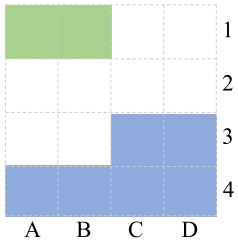
Require: N: enumeration node, *type*: type of check (either “val” for validity, or “ext” for extensibility), $mSup$: support, $mCrd$: cardinality
Ensure: Whether the node satisfies the constraints
1: $validCrd \leftarrow |N.I| \geqslant mCrd$ ▷ Cardinality check
2: $nonRed \leftarrow sup(N.I) = N.CT$ ▷ Redundancy check
3: **if** *type* = “val” **then**
4: $validSup \leftarrow IsSupported(N.CT, mSup)$ ▷ Support and shape check
5: **else if** *type* = “ext” **then**
6: $validSup \leftarrow IsSupported(N.CT \cup N.RT, mSup)$
▷ Support and shape check on the best hypothesis of N's subtree
7: **return** $validCrd \wedge nonRed \wedge validSup$

The transaction is then added to the covered ones (Line 12) and consequently removed from RT_{next} (Line 13) which is assigned to the new node as the set of remaining transactions (Line 14). Finally, the current node is queued for extension (Line 15). When the entire set of remaining transactions is exhausted, the extended nodes potentially entailing co-movement patterns are returned (Line 16).

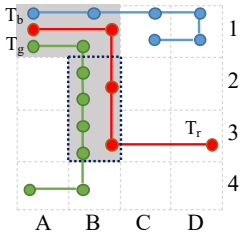
5.2. Validity check and enumeration pruning

CTM relies on three different checks to validate co-movement patterns and to prune subtrees. Checks are implemented through the $Check()$ function.

Cardinality check. This check verifies whether the $\bar{\mathcal{Q}}$-itemset comprises



(a) (Un)connected components.



(b) Redundancy.

Figure 5: Shape and redundancy checks

enough items (i.e., trajectories). The minimum cardinality is necessary to validate a pattern. Additionally, since it holds the anti-monotone property, it can be used to prune N's subtree as well. In particular, an itemset's cardinality monotonically decreases with extensions since itemsets related to nodes in N's subtree are obtained by intersecting $N.I$ with additional transactions (see Algorithm 2 Line 11). Thus, if a $\bar{\mathcal{Q}}$-itemset $N.I$ does not fulfill the cardinality constraint, none of its extensions will, and N's subtree can be pruned.

Support and shape check. This check verifies the minimum support and the connection constraints as in Table 3. The check relies on the $IsSupported(\mathcal{Q}^*, mSup)$ function, which takes in input a set of transactions $\mathcal{Q}^*$ and carries out different checks according to the searched pattern type.

- *Co-location* and *Swarm* patterns do not require shape constraints but only sufficient support ($|\mathcal{Q}^*| \geqslant mSup$).
- *Flow* and *Convoy* patterns require tiles to be connected. This can be checked by verifying if, in $|\mathcal{Q}^*|$, there are transactions corresponding to two tiles connected by a path long at least $mSup$ (see Definition 4). Obviously, if such a connected pair exists, the minimum support constraint is fulfilled too. In our context, the tessellation fits the main memory of every single machine, thus connected components can be found in linear time (Hopcroft and Tarjan, 1973).

Additionally to the shape check discussed above, the $IsSupported()$ function also verifies the minimum support. Notice that this verification is different, depending on whether the pattern is checked for *validity* or for *extensibility* (Algorithm 3 Lines 4 and 6, respectively). Validity is verified on the

current pattern's covered transactions, i.e., $\mathcal{Q}^* = N.CT$. To verify extensibility, we must consider that (differently from cardinality) support and shape hold the monotonic property (as CT is always incremented with new transactions): if a $\bar{\mathcal{Q}}$-itemset does not fulfill the support and shape constraint, some of its extensions can. Thus, the pruning of N's subtree is possible only by verifying that none of the nodes in N's subtree satisfies these constraints. An exact verification would require to enumerating all the nodes in the subtree to generate the corresponding $\bar{\mathcal{Q}}$-itemsets (which is exactly what we want to avoid). We optimize this verification by considering $\mathcal{Q}^* = N.CT \cup N.RT$ as a *best hypothesis*, i.e., an optimistic transaction layout that includes all the transactions in the N's subtree. Even if it has not been verified whether a corresponding $\bar{\mathcal{Q}}$-itemset exists, it serves as an upper-bound to the support and shape check, thus allowing N's subtree to be pruned if the best hypothesis does not pass the check.

Example 6. *With reference to Figure 5a, given $I.CT = \{Q_{A1}, Q_{B1}\}$ (green) and $I.RT = \{Q_{C3}, \ldots, Q_{D4}\}$ (blue), $I.CT \cup I.RT$ cannot produce a convoy with $mSup = 7$ (there are no 7 adjacent tiles) but can potentially produce a convoy with $mSup = 6$ (there are 6 adjacent — blue — tiles).*

Redundancy check. This check avoids generating a $\bar{\mathcal{Q}}$-itemset more than once. The enumeration strategy relies on transaction lexicographic ordering (Algorithm 1 Line 7) to perform a systematic and non-redundant enumeration of transaction combinations. Nonetheless, the same $\bar{\mathcal{Q}}$-itemset I could result from two different sets of transactions S' and S''. We rely on the $\bar{\mathcal{Q}}$-itemset redundancy definition below to prune the enumeration tree.

Definition 10 (Non-redundant enumeration node). *An enumeration node N is non-redundant if $sup(N.I) = N.CT$.*

Example 7 (Redundancy). *With reference to Figure 5b, let N, N' and N'' be enumeration nodes. Let $N.I = \{T_g, T_r\}$ be a $\bar{\mathcal{Q}}$-itemset with $sup(N.I) = \{Q_{A1}, Q_{B1}, Q_{B2}, Q_{B3}\}$ (gray area). The $\bar{\mathcal{Q}}$-itemset can be generated by intersecting $N'.I = \{T_b, T_g, T_r\}$ (with $N'.CT = \{Q_{A1}, Q_{B1}\}$ and $N'.RT = \{Q_{C1}, \ldots, Q_{D4}\}$) with the transaction $Q_{B2} = \{T_g, T_r\}$. Alternatively, it can be obtained directly from Q_{B2}, where $N''.I = Q = \{T_g, T_r\}$ with $N''.CT = \{Q_{B2}, Q_{B3}\}$ (dotted area) and $N''.RT = \{Q_{C2}, \ldots, Q_{D4}\}$). N'' will be discarded since redundant, i.e. $N''.CT \subset sup(N''.I) = sup(N.I)$. Note that, even if N'' refers to transaction Q_{B2}, $N''.CT$ is automatically extended to $\{Q_{B2}, Q_{B3}\}$ by Algorithm 2 Line 6.*

Theorem 2 (Uniqueness of non-redundant enumeration nodes). *There do not exist two distinct non-redundant enumeration nodes N and N' such that $N.I = N'.I$.*

See Proof 2 in Appendix A.

Dropping redundant subtrees does not affect the enumeration completeness as proved by the following theorem.

Theorem 3 (Completeness of non-redundant enumerations). *All $\bar{\mathcal{Q}}$-itemsets generated from redundant enumeration nodes are generated from non-redundant enumeration nodes too.*

See Proof 3 in Appendix A.

Note that all the pruning techniques described so far rely on local information only, i.e. information related to the node except for $\mathcal{Q}$, which is assumed to be distributed. In particular, the non-redundancy check comes at the cost of accessing $\mathcal{Q}$ when the support has to be computed, with complexity $O(|\mathcal{Q}|)$.

5.3. Distributed implementation

CTM is independent of the underlying distributed framework. For our implementation we adopted Spark. A Spark application consists of a *driver* running the main function and demanding concurrent computations. Spark acquires resources, called *executors*, to run distributed computations on cluster nodes. Computations are organized in *job*s, i.e., parallel computations consisting of multiple tasks. *Tasks* are work units sent to one executor. Spark allows to broadcast (i.e., to replicate) read-only shared variables and to define centralized variables on the driver program that can be updated by each executor. Spark provides resilient distributed datasets (RDDs), collections of data items partitioned across the cluster nodes that enable the distributed computation on each partition. RDDs can be created from existing centralized collections through the parallelize function. RDDs allow transformations that return new RDDs such as: map (transforms each data items into a new one) and flatMap (similar to map, but allows returning 0, 1 or more elements). Finally, the shuffle mechanism re-distributes data across partitions following a given criterion. For instance, shuffling data items based on their hash values creates partitions containing a *uniformly distributed* number of data items; see the hashing properties in Menezes et al. (1996).

Algorithm 4 CTM distributed

Require: $\mathcal{Q}$: transactions, $mSup$: support, $mCrd$: cardinality
Ensure: $\mathcal{P}_{RDD}$: co-movement patterns

```
1: broadcast(Q)                                   ▷ Make transactions available to executors
2: P_RDD ← parallelize(Q)                         ▷ Create the RDD
3: P_RDD ← P_RDD.map(MapQ(Q))                     ▷ Map transactions to enumeration nodes
4: do
5:     acc ← false                                ▷ Whether an extendable node exists
6:     P_RDD ← P_RDD.shuffle(Hash(N))             ▷ Redistribute the nodes
7:     P_RDD ← P_RDD.flatMap(MapExtend(N))
                                                  ▷ Map nodes to extend them
8: while acc = true                               ▷ Loop until no node is extendable
9: return P_RDD

10: function MapQ(Q)                              ▷ Map a transaction into an enumeration node
11:     N ← newNode()                             ▷ Create a new enumeration node
12:     N.I ← Q                                   ▷ set the Q̄-itemset
13:     N.CT ← {Q}                                ▷ set Covered Transactions
14:     N.RT ← {Q'|Q' ∈ Q, id(Q') > id(Q)}        ▷ set Remaining Transactions
15:     N.extend ← true                           ▷ mark the node for extension
16:     return N                                  ▷ return the node

17: function MapExtend(N)                         ▷ Extend an enumeration node if needed
18:     if N.extend = true then                   ▷ If it potentially produces valid co-mov. patterns
19:         L ← Extend(N)                         ▷ extend it
20:         acc ← acc ∨ ∃N ∈ L s.t. N.extend = true
                                                  ▷ check if at least an extendable node exists
21:         return L                              ▷ and return the new patterns
22:     else                                      ▷ If N already refers to a valid co-movement pattern
23:         return {N}                            ▷ return it
```

Two key features that allow CTM to efficiently extract co-movement patterns in a distributed environment are the locality of constraint checks and breadth-first exploration. Checks involving local information prevent the need to introduce a single centralized bottleneck which would constraint the feasibility of the approach. As to breadth-first exploration, its benefits are twofold. First, it allows the simultaneous computation of all the $\bar{\mathcal{Q}}$-itemsets "at the same level" of the enumeration tree; intuitively, each level of the enumerated tree is distributedly enumerated by a Spark Job (see Figure 6), allowing a massive parallelization of the generation of $\bar{\mathcal{Q}}$-itemsets. Second, in combination with shuffling, breadth-first exploration allows a balanced distribution of $\bar{\mathcal{Q}}$-itemsets across the executors.

Algorithm 4 describes the distributed implementation of CTM with functions MapQ() and MapExtend() being distributed over the executors. In Line 1, $\mathcal{Q}$ is broadcasted to the executors. This allows each executor to locally access $\mathcal{Q}$ in its entirety (e.g., Line 14). In Lines 2–3, an RDD is created and transactions are mapped to $\bar{\mathcal{Q}}$-itemsets. Since only one RDD can be dis-

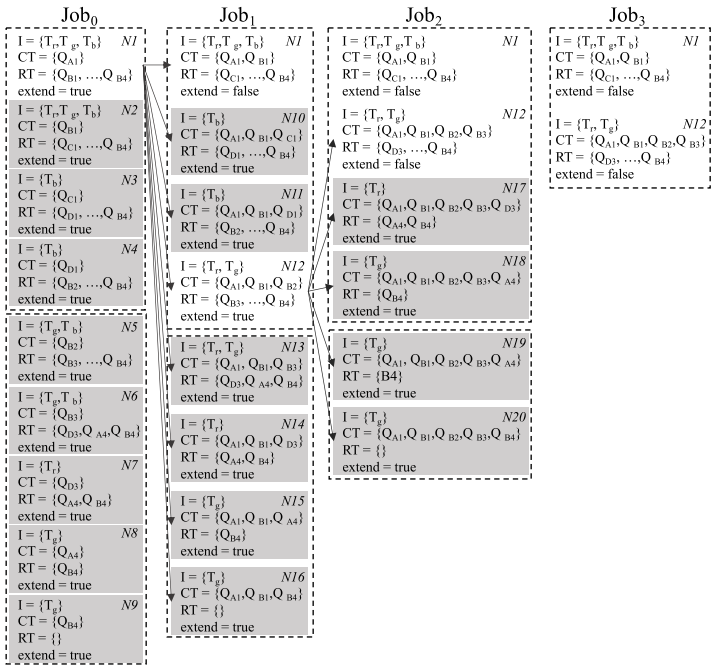


Figure 6: Enumeration tree for co-location patterns in Figure 3 with $mCrd = 2$ and $mSup = 2$: each level of the enumeration tree corresponds to a distributed job.

tributed in Spark, both valid and potential co-movement patterns are stored in the same RDD. To distinguish between co-movement patterns and $\bar{\mathcal{Q}}$-itemsets, we introduce the Boolean flag $N.extend$ which is set to $false$ when the node entails a co-movement pattern (i.e., it must be returned as-is) or to $true$ when it entails a $\bar{\mathcal{Q}}$-itemset that can potentially produce a valid co-movement pattern through further extensions. $\bar{\mathcal{Q}}$-itemsets that cannot lead to a co-movement pattern are discarded in Algorithm 2 Line 4. Obviously, all nodes are initially marked as extendable (Line 15). The while loop from Line 4 to Line 8 uniformly distributes the nodes over the executors and extends each of them (Line 7) independently of the others. The loop iterates until at least a node to be extended exists (i.e., $acc = true$). We omit the distributed version of $Extend()$ since it is marginally affected by distribution. Intuitively, it stores all the patterns in $\mathcal{L}$ and sets $N.extend$ to $false$ when the node entails a valid pattern (i.e., $Check(N, \text{“val”}, mSup, mCrd) = true$) or to $true$ otherwise.

Example 8 (CTM in action). *Figure 6 shows the enumeration tree for*

Table 4: Dataset characteristics

	Milan	Oldenburg	Hermoupolis
MOs	$6.0 \cdot 10^6$	10^6	$4.2 \cdot 10^3$
Points	$2.2 \cdot 10^8$	$6.4 \cdot 10^7$	$5 \cdot 10^7$
Avg traj. points	23 ± 97	65 ± 36	$1.2 \cdot 10^4 \pm 7 \cdot 10^3$
Spatial span	$204km^2$	$634km^2$	$628km^2$
Time span	3 months	246 timestamps	1 week
Dataset type	Real	Synthetic	Synthetic
Spatial feature	Administrative boundaries	Grid	Grid
Temporal feature	Daily time slots	Minutes	Minutes
Additional features	-	-	Transportation, Activity, Speed

co-location patterns in Figure 3 with $mCrd = 2$ *and* $mSup = 2$*. Each level of the tree corresponds to an RDD generated in parallel through a job. RDDs are supposed to be composed by several partitions (dashed rectangles) distributed to executors. Each partition includes several nodes; the grayed ones are non-extendable. Examples of non-extendable nodes are* $N2$ *(redundant to* $N1$*) and* $N3$ *since* $|N3.I| < mCrd$*. The first level of the tree results from mapping of transactions to nodes carried out by* MapQ() *(see Algorithm 4 Line 3). Only the node* $N1$ *satisfies the* $Check()$ *for extensibility and is further expanded. Each of the following levels results from extending in parallel (see Algorithm 4 Line 7) the nodes left over after pruning. Overall, two co-location patterns (*$N1.I = \{T_b, T_g, T_r\}$ *and* $N12.I = \{T_g, T_r\}$*) are found, marked with extend =* $false$*, retained in the RDD, and finally returned.*

6. Evaluation

We show how CTM is applied to several use cases and its performance. We first introduce the datasets (namely Milan, Oldenburg, and Hermoupolis) we use to assess the *effectiveness*, *efficiency*, and *robustness* of our approach. As to effectiveness (Section 6.3), we (i) evaluate how CTM has been (or can be) applied to extract co-movement patterns in real-world (or synthetic) case studies including both geometric and semantic features, and (ii) assess the effects of semantic features on the co-movement patterns. As to efficiency (Section 6.4), we assess the scalability by changing the input parameters of CTM. Finally, we compare CTM and its robustness to related approaches in Section 6.5. All tests run on a cluster of 10 nodes, each equipped with an 8-core i7 CPU@3.60GHz and 16 GB of RAM and interconnected by Gigabit Ethernet. Our implementation is available at `https://github.com/big-unibo/ctm`.

6.1. Dataset description

The datasets differ with respect to MO behaviors and involved features. Their characteristics are summarized in Table 4. Milan is a real trajectory dataset that contains trajectories from $6 \cdot 10^6$ MOs (i.e., individuals) from the Milan metropolitan area (around $200km^2$). Trajectories are sparse in time since they represent inhabitants as well as travelers over three months. Milan produces real-world patterns (e.g., from the main train station to well-known points of interest such as the dome or the stadium). Oldenburg is a synthetic dataset (Brinkhoff, 2002); it contains trajectories from 10^6 MOs from the Oldenburg area (around $600km^2$). Since trajectories span for 246 timestamps, the synthetic trajectories are highly temporally overlapping and condensed in a small period of time. Hence, they are expected to produce co-movement patterns dense in space and time. Hermoupolis is a synthetic dataset too (Pelekis et al., 2015); its trajectories are annotated with additional features and come from $4.2 \cdot 10^3$ MOs from the Athens area (around $600km^2$); trajectories span for 1 week and have an average length of $1.2 \cdot 10^4$ raw points. Additional features range from the means of transport (e.g., bus or bicycle) to the activity undertaken by the user (e.g., sporting, studying, relaxing), and to the speed (e.g., move or stop).

6.2. Parameter tuning

S, $mCrd$, and $mSup$ are dataset- and problem-specific parameters (see Section 2) and have been chosen to answer the following business question: *"Which computable patterns are meaningful for our analysis?"*. To do this we rely on the following guidelines.

1. Choose a tessellation (S) according to the goal of the analysis (e.g., looking for co-movement patterns in the Milan neighborhoods). If a geometric grid is adopted, the tile granularity depends on the level of detail of the analysis. Note that the tessellation should depend on the goal rather than on the optimization of the computation time.

2. Set the number of shared tiles ($mSup$) that is relevant for the analysis (e.g., to be considered as interesting, a co-movement pattern must traverse at least 3 neighborhoods).

3. Set the minimum group cardinality ($mCrd$) that is relevant for the analysis (e.g., if interested in car-sharing applications $mCrd$ could be set between 2 and 6).

Table 5: KPIs by dataset and pattern type

Milan ($mCrd = 100$, $mSup = 12$)

Type	$\|S\|$	Patterns	Enum.	Time (s)	Shape check	Card. check	Red. check
Flow	88	$9.7 \cdot 10^4$	$2.1 \cdot 10^7$	31	53%	82%	30%
Co-loc.	88	$2.3 \cdot 10^5$	$2.6 \cdot 10^7$	44	31%	82%	31%
Convoy[4]	528	0	$5.5 \cdot 10^8$	180	1%	99%	90%
Swarm	528	124	$3.0 \cdot 10^8$	127	8.2%	98%	70%

	Oldenburg ($mCrd = 500$, $mSup = 12$)				Hermoupolis ($mCrd = 400$, $mSup = 12$)			
Type	$\|S\|$	Patterns	Enum.	Time (s)	$\|S\|$	Patterns	Enum.	Time (s)
Flow	90	12	$4.6 \cdot 10^5$	11	230	$9.3 \cdot 10^4$	$2.2 \cdot 10^7$	34
Co-loc.	90	42	$4.6 \cdot 10^5$	14	230	$9.3 \cdot 10^4$	$2.4 \cdot 10^7$	36
Convoy	$1.8 \cdot 10^3$	$1.9 \cdot 10^4$	$2.2 \cdot 10^9$	$1.7 \cdot 10^3$	$1.5 \cdot 10^3$	$1.5 \cdot 10^5$	$1.6 \cdot 10^9$	358
Swarm	$1.8 \cdot 10^3$	$6.5 \cdot 10^4$	$2.4 \cdot 10^9$	$1.8 \cdot 10^3$	$1.5 \cdot 10^3$	$1.0 \cdot 10^7$	$2.7 \cdot 10^9$	$1.1 \cdot 10^3$

4. Verify if the combination of parameters determines meaningful patterns in a reasonable amount of time. If not, iterate on the parameters $mSup$ and $mCrd$ until (i) results are "stable" (i.e., varying $mCrd$ and $mSup$ causes limited changes in the number of co-movement patterns) as prescribed by the elbow method (Satopaa et al., 2011), and (ii) the solution is computable in a reasonable time. While varying the parameters, consider the following.

    - Higher values of $mCrd$ and $mSup$ entail a lower number of co-movement patterns (e.g., fewer groups of MOs will share a longer path) and, in turn, reduce the computation time.
    - The values of $mCrd$ and $mSup$ should also consider the tessellation. For instance, if trajectories are dense, a fine-grained tessellation determines longer patterns. On the other hand, if trajectories are sparse, a fine-grained tessellation amplifies such sparsity reducing the MOs sharing the same tiles; thus, lower values of $mSup$ or $mCrd$ are needed to increase the number of returned patterns.

Although automatizing the approach is beyond the scope of the paper, the points above encode the principles that should drive such an automatic solution.

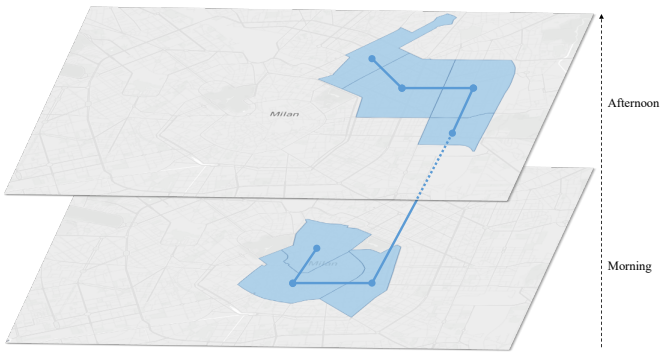


Figure 7: A real-world swarm pattern from the Milan dataset with $mCrd = 100$ and $mSup = 7$.

6.3. Effectiveness

6.3.1. Qualitative evaluation and setup of the tessellation

Table 5 reports the outcomes for all datasets and all co-movement pattern types. To mine co-movement patterns with CTM, it is first necessary to identify the business question and then to properly set the tessellation. Tessellations can be built out of diverse types of features, features can be continuous (e.g., speed) or discrete (e.g., means of transport), absolute (e.g., timestamp) or aggregated (e.g., hour bins), geometric (e.g., latitude/longitude) or semantic (e.g., administrative neighborhoods and municipalities, as in Figure 1).

We provide an example of the swarm patterns extracted on our real case study: Milan. The dataset has been collected during the urban mobility analysis project *"La città intorno"* (`https://lacittaintorno.fondazionecariplo.it/`) that aims to understand the mobility patterns of inhabitants living in suburban neighborhoods. In the context of urban planning, *"La città intorno"* focused on ranking neighborhoods by their attractiveness in order to understand how to allocate economic resources for requalification. The attractiveness of a neighborhood is defined as the percentage of co-movement patterns passing through that neighborhood. To fulfill the analysis, we initially define a tessellation where the spatial feature represents the 88 neighborhoods in Milan (Figure 1a) and the temporal feature represents a relative dimension that partitions absolute timestamps into six bins, such as night (from 0 to 3) and morning (from 8 to 11); overall

[4]Due to the sparsity in time, no convoy pattern is returned in the Milan dataset.

Table 6: An excerpt of attractiveness for neighborhoods in Milan

Neighborhood	Attractiveness
Brera	83%
Duomo	83%
Buenos Aires - Venezia	82%
XXII Marzo	82%
Gallaratese	5%
Lodi - Corvetto	4%
Padova	3%
Adriano	1%

Table 7: Compression of the Milan dataset

Dataset	Raw	Trajectory	Transaction
Cardinality	$2.2 \cdot 10^8$	$2.9 \cdot 10^7$	$3.8 \cdot 10^6$
Size	6 GB	1.4 GB	19 MB

$|S| = 88 \cdot 6 = 528$ tiles. Then, together with domain experts, we set relevant values for $mCrd$ and $mSup$. Figure 7 depicts an example of a swarm pattern for $mCrd = 100$ and $mSup = 7$ in which at least 100 people follow the same path around the city center in the morning and from the city center to the central station in the afternoon. Table 6 shows the results of our attractiveness analysis, highlighting the need for higher requalification in "Lodi - Corvetto", "Padova", and "Adriano"; the most attractive neighborhoods are the ones closest to the city center[5].

We emphasize that while mapping raw trajectory points to tiles causes a loss of (geometric) precision, *no semantic information is lost* as long as the chosen tessellation is adequate for the analysis goal. For instance, losing the precision of a single spatial point (in the order of meters) does not affect the results when looking for groups of trajectories moving through the neighborhoods. Table 7 shows compression obtained by mapping raw trajectory points into the tessellation to obtain the trajectory and the transaction datasets. Cardinality is expressed as raw points, tiles, and trajectory ids depending on CTM's step.

[5]By filtering tiles on the time bin, it is possible to characterize how attractiveness changes during the day.

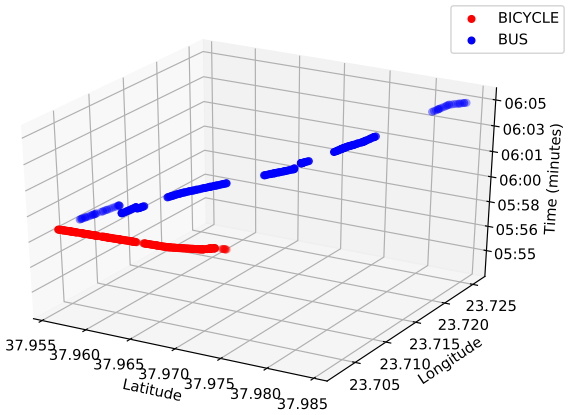


Figure 8: Example of a swarm pattern in the Hermoupolis dataset including space, time, and means of transport (bus and bicycle) features.

6.3.2. Impact of additional features

Since Oldenburg and Hermoupolis are synthetic datasets — and no business value can be extracted from them — we limit the spatial feature to a uniform grid and we shape the size of tiles to get a number of tiles that is comparable to Milan (around 90), then we consider the time granularity of minutes for both of them. Additionally, Hermoupolis contains the following features: means of transport, activity, and speed (i.e., move or stop). These features provide a more precise characterization of co-movement patterns. To highlight the effect of additional features (e.g., transport), Figure 8 shows a qualitative example of a swarm composed of individuals moving together by bicycle and then by bus in the Hermoupolis dataset. When moving by bus (blue), people share longer paths than by bicycle (red), and such paths are also more fragmented since no location can be registered in all adjacent tiles (due to the speed of the bus and the sampling rate of trajectory locations).

It is worth analyzing how much the patterns diverge when adding new features. We tested the Hermoupolis dataset with and without its additional features ($mCrd = 400$, $mSup = 12$). We measured the differences in the co-movement pattern compositions through Adjusted Mutual Information (AMI) (Vinh et al., 2010). The higher (up to 1) the AMI, the higher the number of trajectories shared by co-movement patterns in the tessellations with and without additional features. If the AMI is 1, no information is induced by additional features. Table 8 reports the results, where A, S, and T stand for additional, spatial, and temporal features. Additional features

Table 8: KPIs by pattern type

Type	AMI	ST patterns	STA patterns
Flow	0.17	$5.1 \cdot 10^3$	$9.3 \cdot 10^4$
Co-loc.	0.17	$5.6 \cdot 10^3$	$9.3 \cdot 10^4$
Convoy	0.23	$1.1 \cdot 10^7$	$1.5 \cdot 10^5$
Swarm	0.23	$1.1 \cdot 10^7$	$1.0 \cdot 10^7$

Table 9: Parameters with default values

Param.	Description	Milan	Oldenburg
$mCrd$	Min. group cardinality	100	500
$mSup$	Min. shared path (support)	12	12
$\|S\|$	Tessellation cardinality	528	$1.8 \cdot 10^3$
$\|\mathcal{T}\|$	Trajectories in the dataset	$6.0 \cdot 10^6$	$1.0 \cdot 10^6$
Exec.	Computational units	10	10
RAM	Available RAM per exec.	8 GB	8 GB

sensibly change co-movement patterns as desired. It should be also noted that the number of ST patterns can either increase or decrease with respect to STA patterns. This comes from the combined effect of two phenomena induced by the transition to a finer tessellation: on the one hand there is a proliferation of patterns, on the other hand, patterns could be sparser and, hence, filtered out.

6.4. Efficiency

For the sake of conciseness, we focus on (i) the swarm pattern as it represents the scenario showing the highest computational time, and (ii) Milan and Oldenburg datasets only since Hermoupolis returns results highly similar to Oldenburg. To study the performance in detail, we change the parameters in Table 9, one at a time, and we consider the respective default values for the remaining ones[6].

The impact of pruning on the execution time is summarized in Table 5. For Milan, the table shows the percentage of times each check has been triggered with respect to the enumerated $\bar{\mathcal{Q}}$-itemsets. While percentages are

[6]As to RAM allocation, we assigned each executor 8 GB out of 16 GB since (i) a Spark-related memory overhead is assigned to each executor and (ii) the services responsible for cluster coordination have a non-negligible memory footprint.

Table 10: Enumerated swarm patterns by increasing $mCrd$

Milan			Oldenburg		
$mCrd$	Enum.	Time (s)	$mCrd$	Enum.	Time (s)
25	$7.6 \cdot 10^9$	$1.5 \cdot 10^3$	200	$1.3 \cdot 10^{10}$	$6.0 \cdot 10^3$
50	$1.5 \cdot 10^9$	398	300	$6.6 \cdot 10^9$	$3.7 \cdot 10^3$
75	$5.8 \cdot 10^8$	186	400	$3.7 \cdot 10^9$	$2.6 \cdot 10^3$
100	$3.0 \cdot 10^8$	126	500	$2.4 \cdot 10^9$	$1.8 \cdot 10^3$

Table 11: Enumerated swarm patterns by increasing $mSup$

Milan			Oldenburg		
$mSup$	Enum.	Time (s)	$mSup$	Enum.	Time (s)
6	$4.7 \cdot 10^8$	349	6	$2.7 \cdot 10^9$	$2.2 \cdot 10^3$
8	$3.2 \cdot 10^8$	180	8	$2.4 \cdot 10^9$	$1.8 \cdot 10^3$
10	$3.2 \cdot 10^8$	148	10	$2.4 \cdot 10^9$	$1.8 \cdot 10^3$
12	$3.0 \cdot 10^8$	126	12	$2.4 \cdot 10^9$	$1.8 \cdot 10^3$

here reported independently of each other, a failure of one check prevents the computation of the others. In particular, the failure of the shape or cardinality checks allows to skip the support computation. The cardinality check is always the most effective pruning, preventing the extension of 98% of $\bar{\mathcal{Q}}$-itemsets for swarm patterns. The shape check is more effective for flow patterns than for co-location patterns; extending $\bar{\mathcal{Q}}$-itemsets only within sets of connected tiles allows further pruning.

Tables 10 and 11 show how performance changes by varying the minimum group cardinality (i.e., $mCrd$) and the minimum length of the shared path (i.e., the support $mSup$). For all datasets the number of retrieved patterns and the computational time increase by decreasing the two thresholds. Decreasing $mCrd$ affects CTM more than decreasing $mSup$ due the higher pruning of the cardinality check. For instance, in Milan, halving $mSup$ increases the enumerated nodes from $3.0 \cdot 10^8$ to $4.7 \cdot 10^8$, while halving $mCrd$ increases the enumerated nodes from $3.0 \cdot 10^8$ to $7.6 \cdot 10^9$. Indeed, cardinality checks sensibly affect the pruning (Table 5).

Tables 12 and 13 show how performance changes by varying the cardinality of both the tessellation (i.e., $|S|$) and the trajectories (i.e., $|\mathcal{T}|$) given as input to CTM. As to $|S|$, we achieve finer tessellations by creating finer spatial grids. To do so in the Milan dataset, we approximate the neighborhoods with a spatial grid. For all datasets, we start with a grid of cells with side $2km$ and we incrementally reduce the cell side by 200 meters. Changing the tessellation cardinality $|S|$ affects CTM in two ways. On the one

Table 12: Performance for increasing tiles $|S|$

Milan			Oldenburg		
$\|S\|$	Enum.	Time (s)	$\|S\|$	Enum.	Time (s)
357	$3.0 \cdot 10^8$	126	1791	$2.4 \cdot 10^9$	$1.8 \cdot 10^3$
534	$9.6 \cdot 10^8$	288	1838	$2.5 \cdot 10^9$	$1.9 \cdot 10^3$
679	$1.4 \cdot 10^9$	399	2101	$2.2 \cdot 10^9$	$1.6 \cdot 10^3$
873	$2.0 \cdot 10^9$	647	2232	$2.8 \cdot 10^9$	$2.0 \cdot 10^3$

Table 13: Performance for increasing trajectories $|\mathcal{T}|$

Milan			Oldenburg		
$\|\mathcal{T}\|$	Enum.	Time(s)	$\|\mathcal{T}\|$	Enum.	Time(s)
$1.5 \cdot 10^6$	$1.6 \cdot 10^7$	10	$2.5 \cdot 10^5$	$5.7 \cdot 10^7$	43
$3.0 \cdot 10^6$	$6.2 \cdot 10^7$	22	$5.0 \cdot 10^5$	$4.1 \cdot 10^8$	273
$6.0 \cdot 10^6$	$3.0 \cdot 10^8$	126	$1.0 \cdot 10^6$	$2.4 \cdot 10^9$	$1.8 \cdot 10^3$

hand, increasing the cardinality affects the sparsity of the dataset. Having more tiles fragments trajectory groups until only trajectories matching *exact* points (e.g., latitude and longitude) can be grouped. On the other hand, it broadens the search space of the algorithm. These two effects compensate each other, allowing CTM to explore a search space in the order of thousands of tiles. As to $|\mathcal{T}|$, we achieve smaller datasets by halving the number of trajectories twice; decreasing the trajectory cardinality $|\mathcal{T}|$ sensibly reduces the enumerated space and the computational time: fewer trajectories are less likely to create co-movement patterns.

Finally, Table 14 shows how the amount of RAM and parallelism (i.e., the number of executors) affect CTM. As shown for all datasets, our approach can be applied even when a small amount of RAM is given to each executor. This is due to our implementation on Spark RDDs, where data is split into partitions that are loaded into main memory only when they are ready to be processed. As to the parallelism, reducing the number of executors sensibly affects performance. When CTM runs on a single executor, we move from 2 to 13 minutes in Milan and from 30 minutes to almost 4 hours in Oldenburg. The number of executors affects time (almost) linearly, proving that CTM uniformly distributes the workload among the executors. Note that changing RAM and number of executors does not affect the result in terms of the generated co-movement patterns but only the time necessary for their

Table 14: Time (sec.) for increasing RAM and executors

RAM	Milan	Oldenburg
2 GB	131	$1.8 \cdot 10^3$
4 GB	122	$1.9 \cdot 10^3$
6 GB	122	$1.7 \cdot 10^3$
8 GB	126	$1.8 \cdot 10^3$

Executors	Milan	Oldenburg
1	849	$1.3 \cdot 10^4$
3	297	$4.6 \cdot 10^3$
6	172	$2.6 \cdot 10^3$
10	126	$1.8 \cdot 10^3$

extraction.

We proved that CTM can work in the order of thousands of tiles (Table 5). However, it is unfeasible to provide fixed boundaries in which the method can be applied since they depend on the following factors.

- The distribution of trajectories. Intuitively, 10^4 tiles with highly sparse trajectories can produce fewer co-movement patterns than 10^2 tiles with dense trajectories.
- The pruning effects of $mSup$ and $mCrd$ (Table 5). Even out of a huge number of tiles, the higher is $mCrd$ the easier it becomes to discard *a-priori* tiles with insufficient amounts or trajectories.
- The combination of the above: given a certain dataset, choosing finer tessellations (i.e., tessellations with an increasing number of tiles) affects the cardinality and support of the extracted co-movement patterns (e.g., smaller tiles will contain fewer trajectories requiring a different value of $mCrd$ to extract meaningful co-movement patterns).

6.5. Comparison

We test CTM against SPARE (Fan et al., 2016) (the only big-data approach to the extraction of *generic* co-movement patterns), and PFPGrowth (Li et al., 2008) (a big data approach to Frequent Itemset Mining).

Before diving into the comparison, we first list the main features distinguishing CTM and SPARE.

- CTM allows the extraction of spatial patterns as well as patterns characterized by custom features (e.g., time, speed, or age) of which a discussion has been presented at the end of Section 4. SPARE extracts *only* spatio-temporal patterns and does not provide support to additional features.

- The temporal feature is optional in CTM and can have *custom* semantics, allowing, for instance, the extraction of co-movement patterns in weeks, holidays, or absolute hour bins. SPARE requires a mandatory *absolute* temporal feature.

- SPARE treats the temporal and spatial features sequentially. At first, it groups trajectory points by absolute time bins (e.g., 3 Mar 2020, 10:00:00 and 3 Mar 2020, 11:00:00), then it clusters trajectories in each time bin, and, finally, it uses an Apriori-like approach to create the co-movement patterns out of the clusters from different time bins. This results in the lack of possibility to find co-movement patterns within the same time bin. As a result, if the time bin is coarse (e.g., morning or afternoon) a sensible amount of co-movement patterns is lost (e.g., individuals that move from one side of Milan to the other in a hour).

- Due to their formulation, CTM and SPARE are complementary. CTM allows the retrieval of large groups of trajectories from a coarse tessellation (which justifies the adoption of a bread-first row-enumeration approach), while SPARE allows the retrieval of smaller groups from a finer path (which justifies the adoption of an Apriori-like enumeration). As a result, when SPARE tries to extract huge groups of trajectories, it easily ends up in a memory fault.

While SPARE does not consider additional features, Frequent Itemset Mining approaches could be leveraged to compute *all* FIs and then filter out only the actual co-movement patterns through a post-processing phase[7]. We leverage PFPGrowth, a well-known implementation of distributed FIM already implemented in the Spark suite, to compute *all* the (potential) swarm patterns on the entire Oldenburg dataset. In PFPGrowth the computation of FIs does not rely on Apriori enumeration.

We tested SPARE against the whole Oldeburg, Hermoupolis, and Milan datasets and we stopped computations longer than 10^4 seconds. SPARE failed to compute due to the exponentiality of Apriori enumeration. The points above explain why, by construction, it is unfeasible to have exactly the same co-movement patterns unless the time bin is fine-grained enough to guarantee that no trajectory has more than one location in the same bin. With this in mind, we tried to make the comparison as fair as possible by:

[7]Note that any algorithm for FIM can be picked

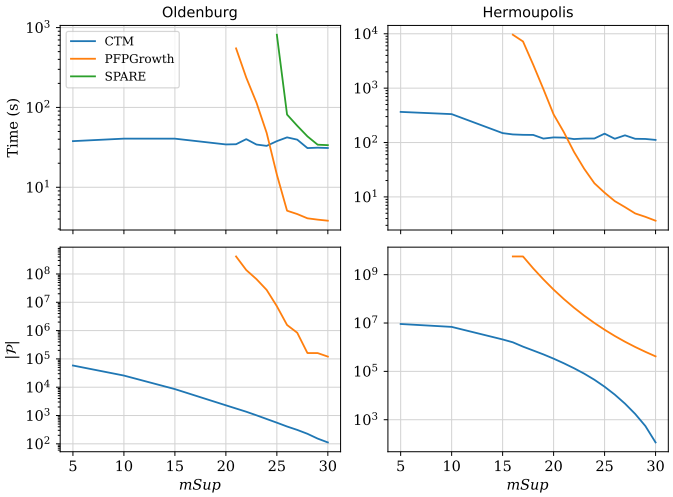


Figure 9: Comparing CTM, SPARE, and PFPGrowth in terms of computational time (top) and co-movement patterns (bottom).

- Oldenburg: the time bin granularity is set in the order of seconds so that no trajectory has more than one location in the same time bin; this increased the size of the tessellation to $|S| = 3422$. We sampled only 2000 trajectories out of 10^6 (i.e., $|T| = 2000$); this is necessary since SPARE follows an Apriori enumeration that is exponential in the number of trajectories in the co-movement patterns.

- Hermoupolis: we dropped the additional semantic features since SPARE is limited to an absolute time dimension and we sampled only 200 trajectories out of $4.2 \cdot 10^3$ (i.e., $|T| = 200$).

- Milan is too sparse in time to produce meaningful results at a fine time-space granularity and with a limited amount of trajectories.

Figure 9 depicts the comparison of the three approaches in terms of computational times and number of retrieved patterns (with the same hardware and software configurations).

- CTM and SPARE retrieve the same patterns (i.e., their $|P|$ lines are overlapping). However, following an Apriori-like strategy, SPARE performance highly depends on the minimum support: the lower it is, the higher the likelihood of extracting patterns with sufficient cardinality

and the wider the portion of search space to be enumerated. This results in memory faults when $mSup$ is below 25 even in the modified Oldenburg dataset and the computation is stopped after three hours in the modified Hermoupolis dataset already for $mSup = 30$ (there is no green line in the top-right chart). Conversely, in CTM the computational time is less affected by the minimum support.

- As to PFPGrowth, although both FIs and co-movement patterns decrease by increasing the $mSup$ threshold (as expected, the higher $mSup$ the lower the valid patterns), the number of FIs to post-process remains orders of magnitude higher than the co-movement patterns, and with lower supports their extraction becomes orders of magnitude slower (Lucchese et al., 2006). This makes FIM approaches strongly inefficient.

We close this section with a remark on the robustness of the algorithm. The tests shown above for CTM on different datasets and its comparison against different algorithms show that its performance is stable and is not the result of overfitting to a specific dataset or configuration.

7. Conclusion

We introduced CTM, a big-data approach to extract spatial and spatio-temporal mobility patterns possibly enriched by additional trajectory features that characterize behavioral mobility patterns. With respect to the existing literature, CTM is general-purpose as it provides a unifying approach to extract different pattern types. CTM is particularly suited for applications characterized by a high number of trajectories to be analyzed on a feature space with limited cardinality (e.g., to capture the daily commuting of a citizen through different neighborhoods, rather than analyzing her detailed path at the single street level). Tests show that in this context CTM is by far more efficient and expressive than previous general-purpose approaches. As new research directions, we plan to: (i) introduce a definition of *group cohesion* to further prune mobility patterns based on the shared tiles, (ii) investigate how the extracted mobility patterns can be summarized in a more succinct representation, (iii) investigate how gridding affects the stability of co-movement patterns, and (iv) consider a unifying extraction of mobility patterns from streaming trajectory data in order to apply CTM to online location-based systems.

Appendix A. Proofs of theorems

Proof 1 (Closeness of $\bar{\mathcal{Q}}$-itemset). *We show by contradiction that (i) all $\bar{\mathcal{Q}}$-itemsets are closed, and (ii) all closed itemsets are $\bar{\mathcal{Q}}$-itemsets. As to (i), if a $\bar{\mathcal{Q}}$-itemset I is not closed, by definition there exists an itemset $I' \supset I$ such that $\sup(I') = sup(I)$; in this case, items in $I' \backslash I$ would be shared by all transactions in $\bar{\mathcal{Q}}$, contradicting the definition of $\bar{\mathcal{Q}}$-itemset. As to (ii), if a closed itemset I is not a $\bar{\mathcal{Q}}$-itemset, there exists an item $t \notin I$ shared by all transactions in $\bar{\mathcal{Q}}$, but in this case $sup(I \cup \{t\}) = sup(I)$, contradicting the closeness definition.*

Proof 2 (Uniqueness of non-redundant enumeration nodes). *The proof follows two statements: (i) lexicographic ordering in CTM guarantees that enumeration sequences are unique, thus $N.CT \neq N'.CT$ even if $N.I = N'.I$, and (ii) every $\bar{\mathcal{Q}}$-itemset I is generated by intersecting transactions belonging to its support, which is $N.CT \subseteq sup(I)$. From (i) and (ii) follows that there exists only one enumeration such that $N.CT = sup(I)$.*

Proof 3 (Completeness of non-redundant enumerations). *If N' is redundant there must exist a non-redundant node N, such that $N.I = N'.I$ and $N.CT \supset N'.CT$. More precisely, according to Algorithm 1 Line 7, transactions in $N.CT$ that are not in $N'.CT$ must have lower ids than those in $N'.CT$. Formally, $N.CT = N'.CT \cup \mathcal{Q}^*$ where, $\forall(Q^*, Q')$ with $Q^* \in \mathcal{Q}^* \subseteq \mathcal{Q}$ and $Q' \in N'.CT$, it is $id(Q^*) < id(Q')$. According to the CTM enumeration strategy $N.RT \supseteq N'.RT$. Thus, every extension obtained from N' will be obtained extending N too.*

Agarwal, P. K., Fox, K., Munagala, K., Nath, A., Pan, J., Taylor, E., may 2018. Subtrajectory clustering: Models and algorithms. In: Proc. of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems. ACM, Houston, Texas, USA, pp. 75–87.

Agrawal, R., Srikant, R., 1994. Fast algorithms for mining association rules in large databases. In: Proc. of VLDB. Vol. 25. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, p. 487–499.

Ailin, H., Zhong, L., Dechao, Z., mar 2019. Movement pattern extraction based on a non-parameter sub-trajectory clustering algorithm. In: Proc. of ICBDA
. IEEE, Suzhou, China, pp. 5–9.

Apiletti, D., Baralis, E., Cerquitelli, T., Garza, P., Pulvirenti, F., Michiardi, P., dec 2017. A parallel mapreduce algorithm to efficiently support itemset mining on high dimensional data. Big Data Research 10, 53–69.

Aung, H. H., Tan, K.-L., 2010. Discovery of evolving convoys. In: Gertz, M., Ludä
scher, B. (Eds.), Scientific and Statistical Database Management. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 196–213.

Bao, X., Lu, J., Gu, T., Chang, L., Xu, Z., Wang, L., nov 2022. Mining non-redundant co-location patterns. IEEE Trans. Neural Networks and Learning Systems 33 (11), 6613–6626.

Bouttier, J., Di Francesco, P., Guitter, E., jul 2003. Geodesic distance in planar graphs. Nuclear physics B 663 (3), 535–567.

Brinkhoff, T., 2002. A framework for generating network-based moving objects. GeoInformatica 6 (2), 153–180.

da Silva, T. L. C., Lettich, F., de Macêdo, J. A. F., Zeitouni, K., Casanova, M. A., 2020. Online clustering of trajectories in road networks. In: Proc. of MDM. IEEE, Versailles, France, pp. 99–108.

Ding, X., Chen, L., Gao, Y., Jensen, C. S., Bao, H., 2018. Ultraman: a unified platform for big trajectory data management and analytics. In: Proc. of VLDB.
Vol. 11. pp. 787–799.

Fan, Q., Zhang, D., Wu, H., Tan, K., 2016. A general and parallel platform for mining co-movement patterns over large-scale trajectories. In: Proc. of VLDB. Vol. 10. pp. 313–324.

Fonseca-Galindo, J. C., de Castro Surita, G., Neto, J. M., de Castro, C. L., Lemos, A. P., 2022. A multi-agent system for solving the dynamic capacitated vehicle routing problem with stochastic customers using trajectory data mining. Expert Systems with Applications 195, 116602.

Francia, M., Gallinucci, E., Golfarelli, M., Santolini, N., dec 2020a. Dart: De-anonymization of personal gazetteers through social trajectories. J. of Inf. Security and Applications 55, 102634.

Francia, M., Gallinucci, E., Vitali, F., may 2019. Map-matching on big data: a distributed and efficient algorithm with a hidden markov model. In: Proc. of MIPRO. IEEE, Opatija, Croatia, pp. 1238–1243.

Francia, M., Golfarelli, M., Rizzi, S., may 2020b. Summarization and visualization of multi-level and multi-dimensional itemsets. Inf. Sciences 520, 63–85.

Fu, Z., Tian, Z., Xu, Y., Zhou, K., 2017. Mining frequent route patterns based on personal trajectory abstraction. IEEE Access 5, 11352–11363.

Gudmundsson, J., van Kreveld, M. J., 2006. Computing longest duration flocks in trajectory data.
In: 14th ACM
Int. Symposium on Geographic Inf. Systems. ACM, Arlington, Virginia, USA, pp. 35–42.

Han, B., Liu, L., Omiecinski, E., jun 2012. NEAT: road network aware trajectory clustering. In: 2012 IEEE 32nd Int. Conf. on Distributed Computing Systems. IEEE, Macau, China, pp. 142–151.

Han, B., Liu, L., Omiecinski, E., 2015. Road-network aware trajectory clustering: Integrating locality, flow, and density. IEEE Trans. Mob. Comput. 14 (2), 416–429.

Han, B., Liu, L., Omiecinski, E., may 2017. A systematic approach to clustering whole trajectories of mobile objects in road networks. IEEE Trans. Knowl. Data Eng. 29 (5), 936–949.

Helmi, S., Banaei-Kashani, F., 2020. Multiscale frequent co-movement pattern mining. In: Proc. of ICDE. IEEE, pp. 829–840.

Hopcroft, J., Tarjan, R., 1973. Algorithm 447: efficient algorithms for graph manipulation. Communications of the ACM 16 (6), 372–378.

Hu, C., Kang, X., Luo, N., Zhao, Q., 2015. Parallel clustering of big data of spatio-temporal trajectory. In: 11th Int. Conf. on Natural Computation, ICNC 2015, Zhangjiajie, China, August 15-17, 2015. pp. 769–774.

Jeung, H., Yiu, M. L., Zhou, X., Jensen, C. S., Shen, H. T., aug 2008. Discovery of convoys in trajectory databases. In: Proc. of VLDB. Vol. 1. VLDB Endowment, pp. 1068–1080.

Jin, F., Hua, W., Francia, M., Chao, P., Orlowska, M. E., Zhou, X., 2023. A survey and experimental study on privacy-preserving trajectory data publishing. IEEE Trans. Knowl. Data Eng. 35 (6), 5577–5596.
https://doi.org/10.1109/TKDE.2022.3174204

Jin, F., Hua, W., Zhou, T., Xu, J., Francia, M., Orlowska, M. E., Zhou, X., sep 2022. Trajectory-based spatiotemporal entity linking. IEEE Trans. Knowl. Data Eng. 34 (9), 4499–4513.
https://doi.org/10.1109/TKDE.2020.3036633

Koh, Y. S., Ravana, S. D., may 2016. Unsupervised rare pattern mining: A survey. TKDD 10 (4)
, 1–29.

Kranstauber, B., Cameron, A., Weinzerl, R., Fountain, T., Tilak, S., Wikelski, M., Kays, R., jun 2011. The movebank data model for animal tracking. Environ. Model. Softw. 26 (6), 834–835.

Kumar, D., Wu, H., Rajasegarar, S., Leckie, C., Krishnaswamy, S., Palaniswami, M., nov 2018. Fast and scalable big data trajectory clustering for understanding urban mobility. IEEE Trans. Intelligent Transportation Systems 19 (11), 3709–3722.

Kwakye, M. M., jul 2020. Conceptual model and design of semantic trajectory data warehouse. Int. J. Data Warehous. Min. 16 (3), 108–131.

Lee, J., Han, J., Whang, K., jun 2007. Trajectory clustering: a partition-and-group framework. In: Proc. of SIGMOD. ACM, Beijing, China, pp. 593–604.

Li, H.,
Wang, Y., Zhang, D., Zhang, M., Chang, E. Y., oct 2008. Pfp: parallel fp-growth for query recommendation. In: Proc. of RecSys. ACM, New York, NY, USA, pp. 107–114.

Li, K., Wang, H., Chen, Z., Chen, L., 2023. Relaxed group pattern detection over massive-scale trajectories. Future Generation Computer Systems 144, 131–139.

Li, Y., Bailey, J., Kulik, L., nov 2015. Efficient mining of platoon patterns in trajectory databases. Data Knowl. Eng. 100, 167–187.

Li, Z., Ding, B., Han, J., Kays, R., 2010. Swarm: mining relaxed temporal moving object clusters. In: Proc. of VLDB. Vol. 3. pp. 723–734.

Liu, Y., Dai, H., Li, B., Li, J., Yang, G., Wang, J., oct 2021. ECMA: An efficient convoy mining algorithm for moving objects. In: Proc. of CIKM. ACM, Queensland, Australia, pp. 1089–1098.

Lu, Y., Sun, Y., Xu, G., Liu, G., 2005. A grid-based clustering algorithm for high-dimensional data streams. In: Advanced Data Mining and Applications, First International Conference. Wuhan, China, pp. 824–831.

Lucchese, C., Orlando, S., Perego, R., 2006. Fast and memory efficient mining of frequent closed itemsets. IEEE Trans. Knowl. Data Eng. 18 (1), 21–36.

Lv, M., Chen, L., Chen, T., Zeng, D., Cao, B., feb 2019. Discovering individual movement patterns from cell-id trajectory data by exploiting handoff features. Information Sciences 474, 18–32.

Menezes, A., van Oorschot, P. C., Vanstone, S. A., dec 1996. Handbook of Applied Cryptography. CRC Press.

Orakzai, F., Calders, T., Pedersen, T. B., 2019. k/2-hop: Fast mining of convoy patterns with effective pruning. In: Proc. of VLDB. Vol. 12. pp. 948–960.

Orakzai, F., Pedersen, T. B., Calders, T., feb 2021. Distributed mining of convoys in large scale datasets. GeoInformatica 25 (2), 353–396.

Pan, F., Cong, G., Tung, A. K. H., Yang, J., Zaki, M. J., 2003. Carpenter: finding closed patterns in long biological datasets. In: Proc. of KDD. ACM, Washington, DC, USA, pp. 637–642.

Parsons, L., Haque, E., Liu, H., et al., 2004. Evaluating subspace clustering algorithms. In: Workshop on Clustering High Dimensional Data and its Applications, SIAM Int. Conf. on Data Mining. Vol. 6. Citeseer, pp. 48–56.

Pei, J., Han, J., Mao, R., aug 2000. CLOSET: an efficient algorithm for mining frequent closed itemsets. In: Proc. of SIGMOD. ACM, Dallas, Texas, USA, pp. 21–30.

Pelekis, N., Sideridis, S., Tampakis, P., Theodoridis, Y., may 2015. Hermoupolis: a semantic trajectory generator in the data science era. ACM SIGSPATIAL Special 7 (1), 19–26.

Phan, N., Poncelet, P., Teisseire, M., sep 2016. All in one: Mining multiple movement patterns. Int. J. of Inf. Technology and Decision Making 15 (5), 1115–1156.

Qiu, M., Pi, D., 2016. Mining frequent trajectory patterns in road network based on similar trajectory. In: Proc. of IDEAL. Springer, Yangzhou, China, pp. 46–57.

Rakthanmanon, T., Campana, B.
J. L., Mueen, A., Batista, G. E. A. P. A., Westover, M. B., Zhu, Q., Zakaria, J., Keogh, E. J., aug 2012. Searching and mining trillions of time series subsequences under dynamic time warping. In: Proc. of KDD. ACM, Beijing, China, pp. 262–270.

Satopaa, V., Albrecht, J. R., Irwin, D. E., Raghavan, B., 2011. Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In: 31st IEEE International Conference on Distributed Computing Systems Workshops. IEEE, Minneapolis, Minnesota, USA, pp. 166–171.

Sim, D., Kwon, O., Park, R., mar 1999. Object matching algorithms using robust hausdorff distance measures. IEEE Trans. Image Process. 8 (3), 425–429.

Tampakis, P., Pelekis, N., Doulkeridis, C., Theodoridis, Y., 2019. Scalable distributed subtrajectory clustering. In: 2019 IEEE Int. Conf. on Big Data. IEEE, Los Angeles, CA, USA, pp. 950–959.

Tran, V., Wang, L., Zhou, L., 2021. A spatial co-location pattern mining framework insensitive to prevalence thresholds based on overlapping cliques. Distributed and Parallel Databases, 1–38.

Tritsarolis, A., Chondrodima, E., Tampakis, P., Pikrakis, A., Theodoridis, Y., sep 2022. Predicting co-movement patterns in mobility data. GeoInformatica, 1–23.

Tritsarolis, A., Theodoropoulos, G.-S., Theodoridis, Y., 2021. Online discovery of co-movement patterns in mobility data. International Journal of Geographical Information Science 35 (4), 819–845.

Van de Geer, J. P., 1995. Some aspects of Minkowski distance. Leiden University, Department of Data Theory.

Vanahalli, M. K., Patil, N., sep 2019. An efficient parallel row enumerated algorithm for mining frequent colossal closed itemsets from high dimensional datasets. Inf. Sci. 496, 343–362.

Vinh, N. X., Epps, J.
, Bailey, J., 2010. Inf. theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. The J. of Machine Learning Research 11, 2837–2854.

Vitali, G., Francia, M., Golfarelli, M., Canavari, M., jan 2021. Crop management with the IoT: An interdisciplinary survey. Agronomy 11 (1), 181.

Vlachos, M., Gunopulos, D., Kollios, G., 2002. Discovering similar multidimensional trajectories. In: Proc. of ICDE. IEEE, pp. 673–684.

Wang, S., Bao, Z., Culpepper, J. S., Sellis, T., Qin, X., sep 2019. Fast large-scale trajectory clustering. In: Proc. of VLDB.
Vol. 13. VLDB Endowment, pp. 29–42.

Wang, Y., McArthur, D., apr 2018. Enhancing data privacy with semantic trajectories: A raster-based framework for GPS stop/move management. Trans. GIS 22 (4), 975–990.

Yan, Z., Chakraborty, D., Parent, C., Spaccapietra, S., Aberer, K., jun 2013. Semantic trajectories: Mobility data computation and annotation. ACM Transactions on Intelligent Systems and Technology 4 (3), 1–38.

Yang, P., Wang, L., Wang, X., Zhou, L., may 2022. SCPM-CR: A novel method for spatial co-location pattern mining with coupling relation consideration. In: Proc. of ICDE. IEEE, pp. 1503–1504.

Yuan, J., Zheng, Y., Zhang, C., Xie, W., Xie, X., Sun, G., Huang, Y., nov 2010. T-drive: driving directions based on taxi trajectories. In: Proc. of the

18th SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, San Jose, CA, USA, pp. 99–108.

Zaki, F. A. M., Zulkurnain, N. F., dec 2018. RARE: mining colossal closed itemset in high dimensional data. Knowledge-Based Systems 161, 1–11.

Zheng, L., Xia, D., Zhao, X., Tan, L., Li, H., Chen, L., Liu, W., 2018. Spatial–temporal travel pattern mining using massive taxi trajectory data. Physica A: Statistical Mechanics and its Applications 501, 24–41.

Zhu, L., Ma, B., Zhao, X., 2010. Clustering validity analysis based on silhouette coefficient. Journal of Computer Applications 30 (2), 139–141.