Information Systems Frontiers (2023) 25:2005–2028
https://doi.org/10.1007/s10796-022-10344-8

Maximal Mixed-Drove Co-occurrence Patterns

Witold Andrzejewski[1] · Pawel Boinski[1]

Accepted: 20 September 2022 / Published online: 20 October 2022

Abstract
One of the interesting types of spatio-temporal patterns is the Mixed-Drove Co-occurrence Pattern (MDCOP), which represents types of objects frequently located together in both space and time. To discover such patterns, methods based on the well-known Apriori strategy are widely used. They involve determining multi-element MDCOPs by building them up iteratively starting with the two-element patterns and then successively adding another element in each iteration. This approach can be very costly, particularly when the data is dense enough to form patterns of significant size. In this paper, we introduce a definition of a new pattern type called a Maximal Mixed-Drove Co-occurrence Pattern. We also propose a new algorithm MAXMDCOP-Miner, which resigns from popular Apriori strategy of generating candidates and, therefore, can discover long pattern without processing all their subsets. Experiments performed on synthetic and real datasets show that MAXMDCOP-Miner has high performance, in particular for dense datasets or tasks with low user-defined thresholds of spatial or time prevalence.

Keywords Spatial data mining · Co-location patterns · Spatio-temporal data · MDCOPs

1 Introduction

In recent years, we have witnessed massive collection of spatio-temporal information. Thanks to cheap and ubiquitous sensors, huge amounts of data about the positions of various objects are recorded in databases every day. It is impossible to analyze such datasets by hand. Therefore, novel computational techniques must be developed and applied to cope with this task. These methods have their roots in the processing, generally referred to as Spatial Data Mining, in which the aspect of time was ignored.

One of the most interesting types of pattern investigated in Spatial Data Mining is a co-location pattern that has been defined as a subset of spatial features that are frequently located together in a spatial neighborhood (Shekhar & Huang, 2001). The spatial feature can be regarded as a type of spatial object (i.e. an object located in space). We say that a particular spatial object is an instance of a spatial feature. For example, a gas station can be treated as a spatial feature, while the gas station on Leyton Street is an instance of that spatial feature. Co-location Pattern Mining (CPM) consists in searching for sets of spatial features, which instances are frequently located close to each other. A familiar, although very simplified, example of such a pattern is known as mutualism in which two organisms of different species exist in a mutually beneficial relationship. In general, we are interested in relationships among multiple spatial features.

Incorporating a time component into co-location patterns resulted in new types of patterns called spatio-temporal co-locations or co-occurrence patterns. Depending on object types (e.g. points, polygons) and concepts of interest measures, many different types of patterns have been proposed, e.g. Huang et al., 2008; Pillai et al., 2012; Qian et al., 2009. In this paper, we focus on Mixed-Drove Co-Occurrence Patterns (MDCOPs) that represent a set of spatial feature types whose instances are located close to each other in geographic space for a significant fraction of time (Celik et al., 2008). For example, MDCOP can represent a predator-prey relationship resulting from predator behavior, i.e. tracking his prey (not necessarily for the whole time and without interruptions). As time component exists in almost every dataset, co-occurrence patterns can provide useful knowledge in many domains, e.g.

✉ Witold Andrzejewski
witold.andrzejewski@cs.put.poznan.pl

Pawel Boinski
pawel.boinski@cs.put.poznan.pl

[1] Faculty of Computing and Telecommunications, Poznan University of Technology, Piotrowo 2, Poznan 60-965, Wielkopolska, Poland

military - battlefield analysis, ecology/health - monitoring pollution and diseases etc.

Existing methods of MDCOP discovery are based on the iterative expansion of basic patterns using the well-known Apriori strategy originally applied to frequent itemset mining problem (Agrawal & Srikant, 1994). In such an approach, the search space is traversed in breadth-first manner. Therefore, in order to find a specific size k pattern, it is necessary to search all its 2^k subsets in advance. This can be considered an important bottleneck in search for larger patterns or patterns in huge datasets.

This problem has been already considered in the context of CPM. In order to eliminate the expensive Apriori-like generation and testing approach, the concept of maximum spatial co-location was introduced by Wang et al. (2009). In short, a co-location pattern is maximal only if it has no superset co-location. We postulate that the same idea can be applied to the mining of MDCOPs.

In this paper, we introduce and define the novel concept of Maximal Mixed-Drove Co-Occurrence Pattern (MaxMDCOP). In order to reduce the number of processed candidates, we propose to remove the iterative pattern building strategy and use solution based on the discovery of the maximum cliques to generate candidates for MaxMDCOPs.

To avoid costly computation of some co-location pattern measures, required to calculate the potential interestingness of the pattern, we introduced several processing improvements. In addition, to find maximal patterns, the following enhancements were applied: (1) computations of upper and lower bounds for candidate's time prevalence, (2) caching of some results for quick determination whether a co-location is spatially prevalent or not without the participation index computations, (3) the participation index computations of all prefixes of co-location pattern in order to avoid repeating similar work multiple times and (4) compression and reuse of co-location pattern instances while computing participation index. We have implemented our new method and performed experiments on synthetic and real world datasets. The results show that we can achieve significant speedups in comparison to the Apriori-like approach.

The structure of the paper is as follows. In Section 2, we discuss related work on MDCOP discovery. Section 3 provides definitions of concepts used throughout this paper. In Section 4, we explain the new algorithm. The results of the conducted experiments are presented in Section 5, followed by a short summary in Section 6.

2 Related Work

MDCOP mining was inspired by the Co-location Pattern Mining idea introduced two decades ago (Shekhar & Huang, 2001). Since the introduction of co-location patterns, many efficient methods for discovering them have been proposed, e.g. Wang et al., 2009; Yoo & Shekhar, 2006; Andrzejewski & Boinski, 2018. Today, CPM can be considered a well-studied area with many popular co-location mining methods utilizing the Apriori generate and test strategy (Agrawal & Srikant, 1994). With the increase of the number of datasets that are collected both automatically and continuously, the time component started to play an important role in various analysis. At the beginning, incorporating temporal aspects led to new models and patterns such as moving averages, first and second order auto-regression or seasonality (Wei, 2006). The common approach was to integrate time as a new, independent dimension. Of course, this reduces the number of possible analysis as some crucial relationships between time and space can be lost.

In response to this problem, new types of patterns, called spatio-temporal co-occurrence patterns, were introduced with two initial directions of research: detecting uniform groups of moving objects and detecting mixed groups of moving objects. Uniform groups, i.e. groups of animals of the same species, can be treated as flock patterns or moving clusters in general (Li, 2014). In flocks and moving clusters discovery, objects must occur in consecutive time frames. MDCOPs are very different from mentioned patterns. Firstly, they describe a set of mixed object types, and secondly these objects can occur in non–consecutive time frames. The first property, i.e. mixed object types, makes MDCOPs similar to co-location patterns. A trivial approach to cope with time in CPM, involving searching for different sets of patterns for subsequent states of the data (Andrzejewski & Boinski, 2019), might omit potentially useful and interesting patterns hidden in data changes between time moments. Another attempt to incorporate time in CPM focused more on associations among spatially extended objects than on the temporal aspects (Yang et al. 2005).

The first type of pattern that treated time as a special separate component in CPM was presented by Cao et al. (2006). The authors defined a so-called co-location episode as a sequence of co-location patterns over consecutive time slots sharing the same feature (called a common feature).

MDCOPs, which are the main topic of this paper, do not impose such constraints. For MDCOPs, incorporating time results in searching for patterns in which spatial features are spatially prevalent (i.e. instances are located close to each other) for the required, not necessarily consecutive, number of time moments (time prevalence). In contrast to co-location episodes, there is no need to define a common feature. It should be noted that other types of spatio-temporal patterns are inspired by co-location concepts, such as SPCOZs (Qian et al., 2009) or STCOPs (Hamdi et al., 2016) etc.

MDCOPs can be discovered using a naive approach, i.e. by applying one of the algorithms of CPM to each time moment and then by making calculations to determine which co-locations are time prevalent. It is easy to notice that, for some patterns, it is not necessary to perform calculations for all time moments. This observation was utilized by Celik et al. (2008). The authors proposed non-naive algorithms MDCOP-Miner and FastMDCOP-Miner. Both algorithms are Apriori-based. In each iteration they discover all size k spatially prevalent patterns and then apply a time prevalence based filtering to detect MDCOPs. These patterns are used to generate size $k+1$ candidates for MDCOPs. In comparison to MDCOP-Miner, FastMDCOP-Miner uses more advanced filters to reduce the number candidates that cannot meet MDCOP requirements. Wang et al. (2019) have tried to improve the efficiency of MDCOP mining by applying a graph based data structure. Unfortunately, some parts of this solution have not been explained, making it impossible to implement them in the way the authors had planned. Since the algorithm is also based on Apriori and the results presented show only slight improvements compared to FastMDCOP-Miner, we will refer to FastMDCOP-Miner as that method is well defined and is the most popular approach to MDCOP mining.

Nevertheless, all Apriori-based methods can suffer from large number of candidates when seeking long patterns (dense datasets, low thresholds for spatial and time prevalence). Similar problems have been addressed in CPM (Tran et al., 2021; Wang et al., 2009; Yao et al., 2016; Yoo & Bow, 2011), where researchers have applied the concept of maximal co-locations to reduce the number of calculations.

In the context of the presented article, particularly noteworthy is the method proposed by Yao et al. (2016). The authors adopted a maximal clique mining method to generate candidates for maximal co-locations and applied a hierarchical verification to construct a condensed instance tree for storing instances of candidates.

This paper adopts the concept of maximal co-location patterns and defines a maximal MDCOP. We introduce a non-Apriori-based algorithm for efficiently calculating such patterns in spatio-temporal datasets. Our proposed solution is inspired by the maximal clique mining method (Yao et al., 2016) dedicated for CPM, however for efficient finding of maximal MDCOPs we apply completely different structures and supporting algorithms.

3 Definitions

Basic Definitions In our work we concentrate on dimensionless objects. Each object is therefore characterized by its coordinate (which might change over time) and a type, called a *spatial feature* or *feature* for brevity. We denote the set of features as F and assume that a total order $\leq_F$ is defined on this set. Given any feature $f \in F$, its *instance* (i.e. spatial object) is denoted as i^f, while the set of all the feature's instances is denoted as I^f. Finally, the set of all feature instances is denoted as $I = \bigcup_{f \in F} I^f$.

As mentioned earlier, each spatial object has a coordinate. We denote the set of all coordinates as K. The properties of this set are irrelevant w.r.t. to this paper as the presented algorithms will only be based on the neighborhood relation given apriori. We introduce this notation solely for the purpose of defining all the necessary formalisms.

We model a spatial (not a spatio-temporal) dataset as a tuple $S^p = (S, p)$ where S is a subset of I ($S \subseteq I$) and $p : S \rightarrow K$ is a function that associates a coordinate in K with all the objects in S.

Any non-negative and symmetric function of two coordinates, which computes a distance between them is called a *distance function* and is denoted as $dist : K \times K \rightarrow \mathbb{R}^+ \cup \{0\}$. Two objects are *neighbors* if the distance between them is less than or equal to a *neighborhood threshold maxdist*. Based on this we define a *neighborhood relation* $R(S^p, maxdist)$ as a set of all pairs of neighboring objects in the spatial dataset S^p. Formally,

$$R(S^p, maxdist) = \left\{ \left(i_1, i_2\right) \in S \times S : dist(p(i_1), p(i_2)) \leq maxdist \right\} \quad (1)$$

For brevity, in the subsequent text, the relation $R(S^p, maxdist)$ is shortened to R if the arguments stem from the context.

Co-location Pattern Mining Most of the presented work has its roots in the paper (Shekhar & Huang, 2001) on co-location pattern mining. Since some of the concepts introduced there are relevant to this paper as well, we introduce the basic definitions related to co-location pattern mining.

Any non-empty subset C of a set of features F ($C \subseteq F$) is called a *spatial co-location pattern*. Spatial co-location patterns represent set of features whose instances are frequently located in their neighborhoods. Given a spatial co-location pattern C, an instance I^C of such pattern (*spatial co-location instance*) is a set of objects which are neighbors (pairwise) and have features in C. Formally,

$$\forall_{i^{f_1}, i^{f_2} \in I^C} \left(i^{f_1}, i^{f_2}\right) \in R \wedge \left\{ f : i^f \in I^C \right\} = C \wedge |I^C| = |C| \quad (2)$$

In order to mine only interesting co-location patterns, some measure of interestingness is needed. Shekhar & Huang (2001) suggest to mine spatially prevalent co-locations and propose a spatial prevalence measure called a *participation index* (denoted Pi). In order to compute a participation index of a co-location C, *participation ratios* (denoted Pr) of every feature in C must be computed first. Given a spatial dataset S^p, let $\mathbb{I}^C_{S^p}$ be the set of all instances of C in S^p. A participation ratio Pr of feature $f \in C$ is defined as

$$Pr(f, C, S^p) = \frac{\left|\left\{ i^{f_1} : f_1 = f \land i^{f_1} \in I^C \in \mathbb{I}^C_{S^p} \right\}\right|}{\left|\left\{ i^{f_1} : f_1 = f \land i^{f_1} \in S^p \right\}\right|} \quad (3)$$

Participation index is the smallest participation ratio over all features $f \in C$, i.e.

$$Pr(C, S^p) = min\{Pi(f, C, S^p) : f \in C\} \quad (4)$$

Participation index has the antimonotonicity property.

In order to specify which patterns are interesting, a *minimal prevalence threshold minprev* is needed. We say that a co-location C is *spatially prevalent* if its participation index is equal to, or greater than the *minprev* threshold, i.e.

$$Pi(C, S^p) \geq minprev \quad (5)$$

The problem of spatial co-location pattern mining, is a problem of efficiently finding all spatially prevalent co-location patterns in a dataset S^p given minimal prevalence threshold *minprev*. We denote a set of all spatially prevalent co-location patterns of size s as $\mathbb{C}_s(S^p, R)$.

Mixed-Drove Co-Occurrence Pattern Mining In this paper we concentrate on one of the spatio-temporal extensions to co-location pattern mining called Mixed-Drove Co-Occurrence Pattern mining (Celik et al., 2008). Below we give definitions which introduce this problem.

We shall start with extending the definitions from the previous paragraph to incorporate the time domain. Let T be a finite set of time moments (i.e. timestamps). We assume that the state of the world is known only at these time moments. Hence, we define a spatiotemporal ST dataset as a set of pairs $S^p_t = (t, S^p)$ where $t \in T$ and t is a unique identifier of the S^p_t pair in the set, while S^p is some spatial dataset. Exemplary spatiotemporal dataset is shown in Fig. 1.

In this particular spatio-temporal data mining problem, we mine co-location patterns such that they are spatially prevalent most of the time. In order to find such patterns, a *spatial co-location time prevalence* measure is defined. Given a pattern C, the spatial co-location time prevalence of this pattern is a ratio of the number of spatial datasets in ST in which C is prevalent to the number of all spatial dataset in ST. Formally,

$$tprev(C) = \frac{|\{t : (t, S^p) \in ST \land Pi(C, S^p) \geq minprev\}|}{|ST|} \quad (6)$$

The time prevalence is antimonotonic similarly to the participation index. A co-location pattern with time prevalence greater than, or equal to, a *minimal time prevalence threshold mintprev* is called a *Mixed Drove Co-occurence Pattern* or MDCOP in short.

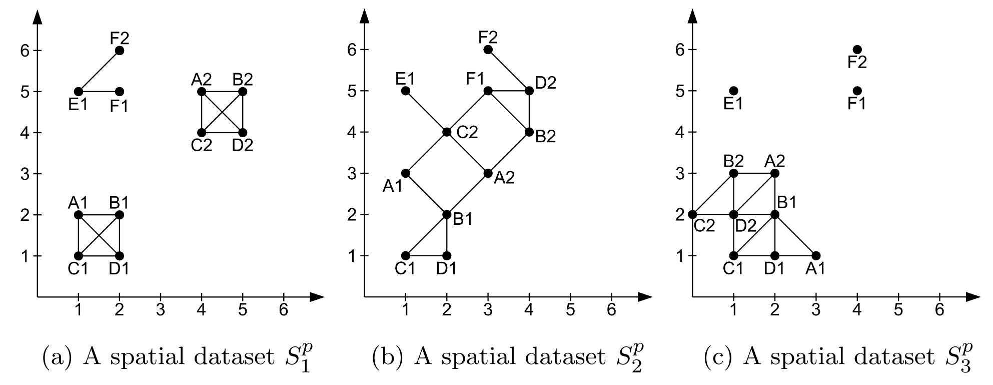
Mixed-Drove Co-Occurrence Pattern mining problem is therefore a problem of efficiently finding all MDCOPs in a spatiotemporal dataset ST given a minimal prevalence threshold *minprev* and a minimal time prevalence threshold *mintprev*. We denote a set of all size s MDCOPs as $\mathbb{C}^T_s(ST)$.

Since the number of MDCOPs can be very large, and many of them are just subsets of the larger ones, mining of all of the MDCOPs is not needed. In many cases mining only the *maximal MDCOPs* is sufficient. A maximal MDCOP is such an MDCOP that no proper superset of it is an MDCOP.

Our paper is devoted to the maximal MDCOP mining problem, which is a problem of efficiently finding all maximal MDCOPs in a spatiotemporal dataset ST given a minimal prevalence threshold *minprev* and a minimal time prevalence threshold *mintprev*. A set of all maximal MDCOPs is denoted as $\mathbb{C}^T(ST)$.

4 MAXMDCOP-Miner

This section contains the main contribution of the paper. We propose a novel algorithm called MAXMDCOP-Miner which is able to efficiently mine maximal MDCOPs. We adopt two parts of the solution presented by Yao et al. (2016) which is devoted to mining maximal spatially prevalent co-locations (not MDCOPS). In particular, we incorporate the idea in which a maximal clique finding algorithm is used



(a) A spatial dataset S^p_1 (b) A spatial dataset S^p_2 (c) A spatial dataset S^p_3

Fig. 1 Exemplary dataset

for generating candidates. We also adapt the co-location instance compression techniques introduced there. However, we substantially extend the ideas from (Yao et al., 2016) to not only increase the compression ratio, but also to be able to reuse some of the computation results multiple times.

On the other hand, we use different data structures. For example, spatial co-location instance identification is done via an iCPI-tree structure (Wang et al., 2009) since it is better suited to support computations of participation index values than InsTable structure presented by Yao et al. (2016). Please refer to (Wang et al., 2009) for methods of construction of such trees. In the following text, we assume that an iCPI-tree is available for every spatial dataset S_t^p in ST. Hence, given a feature instance i^{f_1}, the task of finding all of its neighbors i^{f_2} such that $f_1 < f_2$ can be performed efficiently. The set of such neighbors is denoted $N(i^{f_1}, f_2, S^p, R)$ and formally defined as:

$$N\left(i^{f_1}, f_2, S^p, R\right) = \left\{ i^{f_2} \in S : \left(i^{f_1}, i^{f_2}\right) \in R \wedge f_1 < f_2 \right\} \tag{7}$$

Moreover, we assume that the number of instances $count(f, S_t^p)$ of every feature f in every $S_t^p \in ST$ is known. Formally,

$$count\left(f, S_t^p\right) = \left| \left\{ f : i^f \in S_t^p \right\} \right| \tag{8}$$

Such numbers can be easily found via a single database scan.

In this paragraph, we present a basic idea for MAXMDCOP-Miner algorithm. The detailed description is given in the following sections. The algorithm consists in several steps. Each step produces data required by the next one. Step 1 finds prevalent size 2 co-location patterns in every spatial dataset in ST. In Step 2, a time prevalence of every co-location pattern found in Step 1 is computed and non time prevalent patterns are filtered out. As a result, Step 2 produces all size 2 MDCOPs. In Step 3, using the approach from (Yao et al., 2016), we generate "local" candidates for maximal MDCOPs for every $S_t^p \in ST$. The resulting candidates are sets of features such that every size 2 subset is a) a spatially prevalent size 2 co-location pattern in S_t^p (Step 1) and b) an MDCOP (Step 2). In Step 4, we construct a coherent, global set of candidates for maximal MDCOPs. This is done by processing the results obtained in Step 3. In particular, all duplicates and candidates that cannot be time prevalent are removed. In addition, situations in which candidates are subsets of other candidates are resolved. This processing is non-trivial and cannot be described in few words. Please refer to the appropriate section in the following text for details. In Step 5, the actual mining takes place. Every candidate is verified whether it is time prevalent or not. If not, its subsets are tested. This verification requires computation of the participation index of each candidate. Hence, Step 5 consists of two algorithms: the actual mining algorithm and the participation index computation algorithm.

In the following sections, we give detailed description of each step along with the corresponding algorithm pseudocodes.

4.1 Step 1. Find spatially prevalent size 2 co-location patterns $\mathbb{C}_2(S_t^p, R)$

The aim of the first step is to find prevalent size 2 co-location patterns (denoted $\mathbb{C}_2(S_t^p, R)$) for every spatial dataset S_t^p in ST. These patterns are the basis for computations in subsequent steps.

Step 1 is presented in Algorithm 1. The main loop iterates over each $S_t^p \in ST$ (lines 1–14). To find the prevalent size 2 patterns we scan the iCPI-trees associated with every spatial dataset in ST to retrieve neighbor pairs (i^{f_1}, i^{f_2}) where $f_1 < f_2$ (line 3). The results can be easily grouped by (f_1, f_2) by sorting them (line 4). Each such group corresponds to a spatial co-location $\{f_1, f_2\}$. For each such a co-location (lines 5–13), based on the retrieved pairs, we compute the corresponding participation index (lines 6–9) and compare it with the minimal prevalence threshold *minprev* (line 10). If the result is greater or equal, the co-location $\{f_1, f_2\}$ is added to the result set (line 11).

Example 1 Assuming that the spatiotemporal dataset from Fig. 1 is mined and *minprev* threshold is equal to 0.5, the size 2 spatial co-location patterns for each spatial dataset S_t^p are presented in Table 1 along with their participation indices.

Table 1 Spatial co-location patterns of size 2 $\mathbb{C}_2(S_t^p, R)$

S_1^p				S_2^p				S_3^p	
pattern	*Pi*	pattern	*Pi*	pattern	*Pi*	pattern	*Pi*	pattern	*Pi*
$\{A,B\}$	1	$\{C,D\}$	1	$\{A,B\}$	1	$\{C,D\}$	0.5	$\{A,B\}$	1
$\{A,C\}$	1	$\{E,F\}$	1	$\{A,C\}$	0.5	$\{C,E\}$	0.5	$\{A,D\}$	1
$\{A,D\}$	1			$\{B,C\}$	0.5	$\{C,F\}$	0.5	$\{B,C\}$	1
$\{B,C\}$	1			$\{B,D\}$	1	$\{D,F\}$	0.5	$\{B,D\}$	1
$\{B,D\}$	1			$\{B,F\}$	0.5			$\{C,D\}$	1

Algorithm 1 Find spatially prevalent size 2 co-location patterns

```
Require:
  • a spatiotemporal dataset $ST$
  • a relation $R$ for every $S_t^p \in ST$
  • a minimum prevalence threshold $minprev$
Ensure:
  • a set of size 2 prevalent spatial co-locations $\mathbb{C}_2(S_t^p, R)$ for all $S_t^p \in ST$
1: for $S_t^p \in ST$ do
2:     $\mathbb{C}_2(S_t^p, R) \leftarrow \{\}$
3:     $X \leftarrow$ all neighbours $(i^{f_1}, i^{f_2})$, $f_1 < f_2$ from iCPI-tree for $S_t^p$
4:     Sort $X$ by $(f_1, f_2)$
5:     for each distinct $(f_1, f_2)$ in $X$ do
6:         $A \leftarrow \{(i^{f_x}, i^{f_y}) \in X : f_x = f_1 \wedge f_y = f_2\}$
7:         $a_1 \leftarrow |\{f_1 : (i^{f_1}, i^{f_2}) \in A\}|$
8:         $a_2 \leftarrow |\{f_2 : (i^{f_1}, i^{f_2}) \in A\}|$
9:         $prev \leftarrow min\{a_1/count(f_1, S_t^p), a_2/count(f_2, S_t^p)\}$
10:        if $prev \geq minprev$ then
11:            $\mathbb{C}_2(S_t^p, R) \leftarrow \mathbb{C}_2(S_t^p, R) \cup \{\{f_1, f_2\}\}$
12:        end if
13:    end for
14: end for
```

4.2 Step 2. Find size 2 MDCOPs $\mathbb{C}_2^T(ST)$

In the second step, we aggregate the results obtained in the previous step in order to find a set of all size 2 MDCOPs (denoted $\mathbb{C}_2^T(ST)$). Note that the patterns obtained in Step 1 are candidates for size 2 MDCOPs. In order to determine which of them are time prevalent, it is sufficient to count, for each unique co-location pattern from Step 1, the number of $\mathbb{C}_2(S_t^p, R)$ sets in which it appears.

The second step is presented in Algorithm 2. In a loop (lines 3–7), for every spatial dataset we iterate over every pattern in the corresponding set of patterns obtained in Step 1 (lines 4–6) and count the number of times each of them appears in spatial datasets. In the second loop (lines 8-13), for every unique pattern we compute its time prevalence based on the obtained counter value (line 9). In case the pattern is time prevalent, we add it to the result set (line 11).

Example 2 In this example we continue the mining process from Example 1. We assume that *mintprev* threshold is equal to $\frac{2}{3}$. All distinct prevalent spatial co-location patterns found in Step 1 are presented in Table 2. Given the value of *mintprev* threshold, each such candidate should appear in at least two out of three $\mathbb{C}_2(S_t^p, R)$ sets obtained in Step 1. The co-location patterns in Table 2 are divided into two columns: non time prevalent and time prevalent. By definition, time prevalent ones are spatio-temporal co-location patterns of size 2.

Table 2 Spatiotemporal co-location patterns of size 2 $\mathbb{C}_2^T(ST)$

All distinct co-locations from step 1			
Non time prevalent		time prevalent $\mathbb{C}_2^T(ST)$	
pattern	*tprev*	Pattern	*tprev*
$\{B, F\}$	1/3	$\{A, B\}$	1
$\{C, E\}$	1/3	$\{A, C\}$	2/3
$\{C, F\}$	1/3	$\{A, D\}$	2/3
$\{D, F\}$	1/3	$\{B, C\}$	1
$\{E, F\}$	1/3	$\{B, D\}$	1
		$\{C, D\}$	1

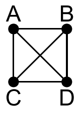
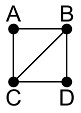
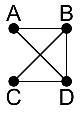
4.3 Step 3. Build local candidates for maximal MDCOPs $K(S_t^p)$ for $S_t^p \in ST$

In this step, we find initial candidates for maximal MDCOPs. In order to do so, we incorporate a slightly modified approach from (Yao et al., 2016). The cited paper presents a method for mining maximal spatial co-location patterns. In that approach, the candidates for such patterns are maximal cliques (obtained via Bron-Kerbosch algorithm with later modifications) in a specially constructed graph $G(V, E)$. In this graph, vertices correspond to features ($V \subseteq F$), while edges correspond to size 2 spatially prevalent co-location patterns ($E = C_2(S_t^p)$). A maximal clique in such a graph is a set of features, where each pair is a spatially prevalent co-location pattern and no more features can be added without

Algorithm 2 Finding size 2 MDCOPs

```
Require:
  • a set of prevalent spatial co-location patterns of size 2 $\mathbb{C}_2(S_t^p, R)$ for every $S_t^p \in ST$
  • a minimum time prevalence threshold $mintprev$
Ensure: a set of time prevalent co-location patterns of size 2 $\mathbb{C}_2^T(ST)$
1: $\mathbb{C}_2(ST) \leftarrow \{\}$
2: $T \leftarrow$ a hash table, indexed by sets, initial value for every entry is 0
3: for $S_t^p \in ST$ do
4:     for $\{f_1, f_2\} \in \mathbb{C}_2(S_t^p, R)$ do
5:         $T[\{f_1, f_2\}] \leftarrow T[\{f_1, f_2\}] + 1$
6:     end for
7: end for
8: for $\{f_1, f_2\} \in$ keys of $T$ do
9:     $tprev \leftarrow T[\{f_1, f_2\}]/|ST|$
10:    if $tprev \geq mintprev$ then
11:        $\mathbb{C}_2(ST) \leftarrow \mathbb{C}_2(ST) \cup \{\{f_1, f_2\}\}$
12:    end if
13: end for
```

Table 3 Local candidates for maximal spatiotemporal co-locations $K(S_t^p)$ for $S_t^p \in ST$

Dataset	$C_2(S_t^p) \cap C_2(ST)$	Graph	Maximal cliques (candidates)
S_1^p	$\{A,B\}, \{A,C\}, \{A,D\}, \{B,C\}, \{B,D\}, \{C,D\}$		$\{A,B,C,D\}$
S_2^p	$\{A,B\}, \{A,C\}, \{B,C\}, \{B,D\}, \{C,D\}$		$\{A,B,C\}, \{B,C,D\}$
S_3^p	$\{A,B\}, \{A,D\}, \{B,C\}, \{B,D\}, \{C,D\}$		$\{A,B,D\}, \{B,C,D\}$

Algorithm 3 Building local candidates for maximal MDCOPs

```
Require:
  • a set of spatially prevalent co-location patterns of size 2 $\mathbb{C}_2(S_t^p, R)$ for every $S_t^p \in ST$
  • a set of time prevalent co-location patterns of size 2 $\mathbb{C}_2(ST)$
Ensure:
  • a set of local candidates for maximal co-locations $K(S_t^p)$ for every $S_t^p \in ST$
1: for $S_t^p \in ST$ do
2:     $E \leftarrow \mathbb{C}_2(S_t^p, R) \cap \mathbb{C}_2(ST)$
3:     $V \leftarrow \{f : f \in \{f_1, f_2\} \in E\}$
4:     $K(S_t^p) \leftarrow$ maximal cliques in graph $G(V, E)$        ▷ Bron-Kerbosch alg.
5: end for
```

breaking this property. Due to the antimonocity of the used prevalence measure, all size 2 subsets of any spatially prevalent co-location pattern must also be spatially prevalent. Hence, a maximal clique in graph $G(V, E)$ is a candidate for a maximal spatially prevalent co-location pattern.

We modify the above approach by limiting the edge set to such co-location patterns that appear in $C_2(S_t^p)$ and are MDCOPs as well ($E = C_2(S_t^p) \cap C_2(ST)$). Thus, the candidates contain only spatially and time prevalent pairs of features. Due to the antimonotonicity of the time prevalence, the candidates found in this step are the biggest co-location patterns that: a) might be MDCOPs and b) can be spatially prevalent in the corresponding spatial dataset.

The pseudocode for Step 3 is presented in Algorithm 3. We iterate over every dataset in ST (lines 1–5). For each such dataset, we create the graph described earlier (lines 2 and 3). Finally, we search for maximal cliques by using the Bron-Kerbosch algorithm (line 4).

Example 3 We continue the Example 2. For each spatial dataset S_t^p we find a set of locally spatially prevalent spatiotemporal co-location patterns $C_2(S_t^p) \cap C_2(ST)$. Those sets are presented in second column of Table 3. We treat those sets as edges in a graph. Each corresponding graph is presented in the third column of the table. We analyze the graphs to find maximal cliques using Bron-Kerbosch algorithm. Maximal cliques are presented in the fourth column of the table.

4.4 Step 4. Build global candidates for maximal MDCOPs $K(ST)$

The candidates obtained in Step 3 must be further refined in order to get a coherent, global set of candidates. This is needed because candidates found in one of the datasets in ST might be subsets of, or be equal to, candidates found in other spatial datasets. This problem can be easily solved by removing duplicates and subsets. However, the candidates found in the previous step might be also not time prevalent. While time prevalence is accurately verified during the subsequent step, we can perform some initial filtering based on the information obtained up to now. Notice, that we can compute an upper bound on the candidate's time prevalence by finding the number of times it appears in $K(S_t^p)$ sets (either equal to or as a subset of another candidate). If the upper bound is less than the *mintprev* threshold, the candidate cannot be time prevalent.

At this moment, we can choose to use either a top-down approach or a bottom-up approach to find global candidates. A top-down approach is very similar to the one used in Step 5 for the actual mining, while bottom-up approach is based on the well-known Apriori algorithm (Agrawal & Srikant, 1994). Below we describe the bottom-up method, which has shown to behave better in our internal experiments. For a top-down approach we refer the reader to a conference version of this paper (Andrzejewski & Boinski, 2021).

The Apriori algorithm can be used to find (sub)sets such that the value of their interestingness measure (e.g. support, participation index) is greater than the specified threshold. The measure itself needs to be antimonotonic. The Apriori algorithm is a generate-and-test approach in which smallest possible candidates for result sets are generated first and then the value of the interestingness measure is computed for each of them. Only candidates with the interestingness greater than the specified threshold are retained. Based on the retained sets, new, larger candidates are generated and the process is repeated. The algorithm ends when no more candidates can be generated. Since the Apriori method generates its own candidates, there might be some ambiguity with the term "candidate". Notice that it may mean a candidate in MAXMDCOP-miner and a candidate in the Apriori method. For the clarity, we call the Apriori candidates "r-candidates".

In this step, we want to use the Apriori to find co-locations that a) have an upper bound on the time prevalence greater than the *mintprev* threshold and b) are maximal. We adapt the algorithm as follows. The initial set of r-candidates is built from the set $\mathbb{C}_2^T(ST)$ of all size 2 MDCOPs found in Step 2. R-candidates one size larger are created by joining two smaller co-locations by their prefix in the same way as in the traditional Apriori (Agrawal & Srikant, 1994) or in the traditional co-location pattern mining (Shekhar & Huang, 2001). As a measure of interestingness we use the upper bound on time prevalence computed based on $K(S_t^p)$ sets as described above. This upper bound is antimonotonic since given two co-locations C and C' such that $C \subset C'$, C can only be subset of more sets in $K(S_t^p)$ for $S_t^p \in ST$ than C'. Finally, during the mining process, all non maximal co-locations are removed if no longer needed. The resulting maximal co-locations are candidates for Step 5 of MAXMDCOP-miner.

The Apriori-based bottom-up algorithm for building global candidates is shown in Algorithm 4. The algorithm stores computed co-locations in sets denoted as K_i', where i is the size of the co-locations. We start by initializing the set K_2' (line 1). Based on this set, a new set of size 3 r-candidates is created (line 2). We also keep *rcsize* variable which stores a current size of r-candidates. Before the Apriori loop starts, a new set K_{rcsize}' (in this case K_3') is initialized (line 4). While the set of r-candidates is not empty, we repeat the main loop (lines 5–16). For each candidate, (line 6) we compute an upper bound of its time prevalence (lines 7–9). If the computed upper bound is greater than the minimal time prevalence threshold, the r-candidate is stored in K_{rcsize}' set (line 10). Once all the r-candidates have been verified, we remove all co-location patterns that became non maximal (line 12). Next, new candidates are created, *rcsize* variable is updated and a new set K'_{rcsize} set is initialized (lines 13–15). Once the main loop ends, the K'_i sets store maximal co-locations such that their upper bound on time prevalence is greater than the *mintprev* threshold. The final $K(ST)$ set is computed as a sum of all K'_i (line 17).

Example 4 In the Example 3 we have found local candidates for maximal spatio-temporal co-locations (see Table 3). We can use them in order to compute an upper bound on candidate's time prevalence. In order to generate all maximal candidates, we use an Apriori-based bottom-up approach presented in Algorithm 4. The initial set of size 3 r-candidates is generated from the candidates in $K'_2 = \mathbb{C}^T_2(ST)$. Four r-candidates are generated:

- $\{A, B, C\}$: join $\{A, B\}$ and $\{A, C\}$, $\{B, C\}$ is in $\mathbb{C}^T_2(ST)$,
- $\{A, B, D\}$: join $\{A, B\}$ and $\{A, D\}$, $\{B, D\}$ is in $\mathbb{C}^T_2(ST)$,
- $\{A, C, D\}$: join $\{A, C\}$ and $\{A, D\}$, $\{C, D\}$ is in $\mathbb{C}^T_2(ST)$,
- $\{B, C, D\}$: join $\{B, C\}$ and $\{B, D\}$, $\{C, D\}$ is in $\mathbb{C}^T_2(ST)$.

For each of the r-candidates we check whether they are a subset of at least one candidate for every S^p_i. Based on the number of such datasets we estimate the upper bound on the time prevalence. The results are presented in Table 4. Since r-candidate $\{A, C, D\}$ cannot be time prevalent, $K'_3 = \{\{A, B, C\}, \{A, B, D\}, \{B, C, D\}\}$. Before the next Apriori iteration we remove all the subsets of sets in K'_3 from K'_2 leaving it empty. Based on the sets in K'_3 we attempt to create new r-candidates. Only r-candidates $\{A, B, C\}$ and $\{A, B, D\}$ can be joined into $\{A, B, C, D\}$, but since $\{A, C, D\}$ is not time prevalent, then $\{A, B, C, D\}$ cannot be time prevalent as well. Hence, the set of global candidates $K(ST) = K'_2 \cup K'_3 = \{\{A, B, C\}, \{A, B, D\}, \{B, C, D\}\}$.

Table 4 Mining global candidates

r-candidate	appears in	upper bound	status
$\{A, B, C\}$	$S^p_1(\{A, B, C, D\})$	$\frac{2}{3}$	possibly time prevalent
	$S^p_2(\{A, B, C\})$		
$\{A, B, D\}$	$S^p_1(\{A, B, C, D\})$	$\frac{2}{3}$	possibly time prevalent
	$S^p_3(\{A, B, D\})$		
$\{A, C, D\}$	$S^p_1(\{A, B, C, D\})$	$\frac{1}{3}$	not time prevalent
$\{B, C, D\}$	$S^p_1(\{A, B, C, D\})$	1	possibly time prevalent
	$S^p_2(\{B, C, D\})$		
	$S^p_3(\{B, C, D\})$		

4.5 Step 5. Mine maximal MDCOPs $\mathbb{C}^T(ST)$

In the last step, we use all the results obtained previously to mine maximal MDCOPs. In order to achieve this result, we

Algorithm 4 Find global candidates for maximal MDCOPs

```
Require:
  • a set of local candidates for maximal MDCOPs $K(S^p_t)$ for every $S^p_t \in ST$
  • a set of MDCOPs of size 2 $\mathbb{C}^T_2(ST)$
  • a minimum time prevalence threshold $mintprev$
Ensure: a set of global candidates for maximal MDCOPs $K(ST)$
1:  $K'_2 \leftarrow \mathbb{C}^T_2(ST)$
2:  $RCS = BuildRCandidateSet(K'_2)$
3:  $rcsize \leftarrow 3$
4:  $K'_{rcsize} \leftarrow \{\}$
5:  while $|RCS| > 0$ do
6:      for $RC \in RCS$ do
7:          $c \leftarrow 0$
8:          for $S^p_t \in ST$ do: if $\exists X \in K(S^p_t) : RC \subseteq X$ then $c \leftarrow c + 1$
9:          $tprev = c/|ST|$
10:         if $tprev \geq mintprev$ then $K'_{rcsize} \leftarrow K'_{rcsize} \cup \{RC\}$
11:     end for
12:     for $C \in K'_{rcsize}$ do: Remove all subsets of $C$ from $K'_{rcsize-1}$
13:     $RCS = BuildRCandidateSet(K'_{rcsize})$
14:     $rcsize \leftarrow rcsize + 1$
15:     $K'_{rcsize} \leftarrow \{\}$
16: end while
17: $K(ST) \leftarrow \bigcup_{i=2,\ldots,rcsize} K'_i$
```

use the top-down approach. We verify whether the candidates obtained in the previous step are really time prevalent or not. If a candidate turns out to not be time prevalent, all of its subsets (one item smaller) are analysed. We augment this basic top-down schema with multiple optimizations in order to reduce the execution time and memory footprint.

To compute the time prevalence, we need to know in which spatial datasets in ST the candidate is spatially prevalent. Unfortunately, the participation index computations are expensive. We approach this problem in two ways. First, we incorporate several optimizations which reduce the required number of spatial prevalence computations. Second, we provide a novel participation index computation algorithm, which uses special data structures to cache and reuse intermediate results of computations to reduce the cost as much as possible. Since this algorithm is rather complex, we describe it separately in the next section (Section 4.6).

The pseudocode of Step 5 is presented in Algorithm 5. The first operation is the initialization of three results caches (line 2). Each cache is a family of data structures, one for each spatial dataset in ST. The lpc_t structures are sets of co-locations that are spatially prevalent at corresponding S_t^p, but are not time prevalent. Since any subset of such a co-location is also spatially prevalent, storing this information allows to skip the participation index computations in some situations. Similarly, the npc_t structures are sets of smallest co-locations in corresponding S_t^p that are known to not be spatially prevalent. If a candidate contains such a co-location, it cannot be spatially prevalent as well. Finally, pc_t structures are associative arrays that map co-location patterns to corresponding participation indices in the corresponding S_t^p (if known). Please note that npc and pc caches are modified only by the participation index computation algorithm described in the next section.

After the caches have been created, the set Y is initialized with maximal MDCOP candidates obtained in Step 4 (line 3). Since we use a top-down approach, candidates are processed from the largest to the smallest. In each iteration of the main algorithm loop (lines 3–37), the largest candidates are moved from the set Y into the set M (line 10). Since new candidates can be generated during mining, a structure for storing them is needed. Thus, a set N is created (line 11).

The candidates from the set M (the largest, still unverified candidates) are processed in the outer for loop (lines 12–34). A candidate co-location pattern currently processed in the outer loop is denoted as P. For each such candidate we keep 4 sets: TP_e, TP_c, TN_e and TN_c (initialized in line 13). The TP sets store time moments t of S_t^p datasets at which the candidate P is known to be spatially prevalent. The TN sets store time moments at which the candidate P is known to not be spatially prevalent. The difference between c and e variants of these sets is that for c sets the candidate's spatial prevalence is determined based on the participation index computations, while for e, the candidate's spatial prevalence stems from results obtained in previous iterations.

Sizes of the TP and TN sets allow to find upper and lower bounds on a candidate's time prevalence. A lower bound $lb(P)$ can be computed by assuming that the candidate is spatially prevalent only in datasets S_t^p for $t \in TP_c \cup TP_e$. Hence,

$$lb(P) = \frac{|TP_e| + |TP_c|}{|ST|}. \tag{9}$$

Similarly, upper bound $ub(P)$ can be computed by assuming that the candidate is spatially prevalent in all datasets S_t^p in ST except for the ones with time moments in $TN_c \cup TN_e$. Thus,

$$ub(P) = 1 - \frac{|TN_e| + |TN_c|}{|ST|}. \tag{10}$$

For each of candidates P, we attempt to determine whether they are time prevalent based on the information obtained up to now. This is done in the first inner for loop (lines 14–19). In each iteration, we check results related to the one spatial dataset S_t^p from ST. We check whether the candidate is a subset of any local candidates in $K(S_t^p)$. The $K(S_t^p)$ set contains all maximal feature sets (co-location patterns) such that they a) might be an MDCOP and b) can be spatially prevalent in the corresponding spatial dataset. If the candidate P is not a subset of one of them, it either cannot be an MDCOP or cannot be spatially prevalent in the corresponding spatial dataset. In either of cases, we know that the current spatial dataset S_t^p cannot contribute to the candidate's time prevalence. Thus, the time moment t is added to the TN_e set (line 15). Second check uses the non prevalent co-locations cache npc_t. As mentioned before, the cache contains the smallest known patterns that are not spatially prevalent. If this set contains any pattern that is a subset of the candidate P, the candidate cannot be spatially prevalent and thus, the time moment t should be added to the TN_e set (line 16). Third and last check makes use of the lpc_t cache introduced earlier. If the candidate P is a subset of a co-location pattern that is spatially prevalent in S_t^p, it must be spatially prevalent as well (due to the antimonotonicity of the participation index measure). In such a case, the time moment t is inserted into the TP_e set (line 17).

Each iteration of the first inner for loop might update either the TP_e or the TN_e set. Hence, with each iteration tighter bounds on candidate P time prevalence can be computed. If $lb(P) \geq mintprev$ (P is time prevalent) or $ub(P) < mintprev$ (P is not time prevalent), no further computations are necessary (line 18). However, if neither of these conditions is fulfilled by the end of the first inner loop, accurate computations are required (line 20).

Accurate computations are performed in the second inner for loop (lines 21–26). This loop iterates over all S_t^p datasets for which the spatial prevalence status of candidate P is not yet known. In each iteration, we compute the candidate's participation index (line 22) using Algorithm 7 described later in Section 4.6. During computations, the npc and pc caches are updated as well. Depending on the result, the time moment of the current spatial dataset is added either to TP_c or TN_c set (lines 23 and 24).

Based on the updated TP_c and TN_c sets, lower $lb(P)$ and upper $ub(P)$ bounds on the candidate's time prevalence are computed. If $lb(P) \geq mintprev$ or $up(P) < mintprev$, further computations can be aborted (line 25).

After the second inner for loop, we check whether the candidate is time prevalent. If it is, the set $\mathbb{C}^T(ST)$ is updated accordingly (lines 28 and 29). If it is not, we add all of the candidate's subsets (one item smaller) to the set N (line 31). Moreover, the lpc_t sets are updated accordingly (line 32).

After the outer for loop ends, all the candidates in M are processed. However, before the next iteration of the main loop can be started, it is necessary to filter out all non maximal candidates in the set N. For this purpose we search $\mathbb{C}^T(ST)$ for supersets of candidates in N. Only candidates without supersets are added to the set Y (line 35) to be processed in the subsequent iteration of the main loop.

The main loop is terminated either when the set Y is empty (line 4) or only size 2 candidates are left (line 6). Note that size 2 MDCOPs were already mined in Steps 1 and 2. Thus, no costly computations are necessary. In order to find maximal size 2 MDCOPs, it is sufficient to compute the intersection $Y \cap \mathbb{C}_2^T(ST)$ (lines 6-8). This also explains the condition in line 32. Since the smallest analysed candidates are size 3, then the lpc_t cache should store only candidates with more than 3 items.

Once the while loop ends, the set $\mathbb{C}^T(ST)$ stores all the maximal MDCOPs.

Example 5 In this example, we refer to three tables: Table 5, 6 and 7. The figures in the row "Data structures" of these tables however are not explained. They will be covered in the example for the participation index computation algorithm in the Section 4.6.

In the Example 4 three global candidates were found: $\{A,B,C\}$, $\{A,B,D\}$ and $\{B,C,D\}$. Since all of the candidates are of the same size, all of them are processed in the same iteration of the main loop of the Step 5. Moreover, let us assume that they are processed in the order given above.

First, the candidate $\{A,B,C\}$ is processed. All the computations regarding this candidate are presented in Table 5. Before the actual participation index computations take place, the pattern is checked against the sets of local candidates and the npc_t and lpc_t caches (see row "Estimating results"). Analysis of S_1^p and S_2^p spatial datasets, does not yield any results. Hence, the TN_e and TP_e sets are empty and

Table 5 Processing of the candidate pattern {A,B,C}

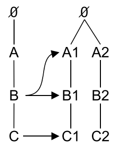
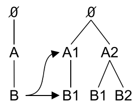
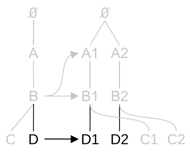
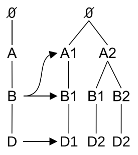
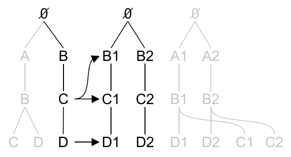
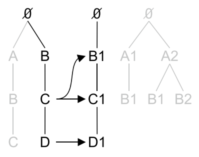
Pattern	$\{A,B,C\}$		
Spatial dataset	S_1^p	S_2^p	S_3^p
Estimating	$TN_e = \{\ \}$	$TN_e = \{\ \}$	$TN_e = \{3\}$
results	$TP_e = \{\ \}$	$TP_e = \{\ \}$	$TP_e = \{\ \}$
	$lb = 0$	$lb = 0$	$lb = 0$
	$ub = 1$	$ub = 1$	$ub = 2/3$
Accurate	$Pi = 1$	$Pi = 0$	–
computations	$TN_c = \{\ \}$	$TN_c = \{2\}$	–
	$TP_c = \{1\}$	$TP_c = \{1\}$	–
	$lb = 1/3$	$lb = 1/3$	–
	$ub = 2/3$	$ub = 1/3$	–
	$pc_1[\{A,B\}] = 1$	$pc_2[\{A,B\}] = 1$	–
	$pc_1[\{A,B,C\}] = 1$	$npc_2 = \{\{A,B,C\}\}$	–
Data structures			–

Table 6 Processing of the candidate pattern {A,B,D}

Pattern	$\{A,B,D\}$		
Spatial dataset	S_1^p	S_2^p	S_3^p
Estimating	$TN_e = \{\ \}$	$TN_e = \{2\}$	$TN_e = \{2\}$
results	$TP_e = \{\ \}$	$TP_e = \{\ \}$	$TP_e = \{\ \}$
	$lb = 0$	$lb = 0$	$lb = 0$
	$ub = 1$	$ub = 2/3$	$ub = 2/3$
Accurate	$Pi = 1$	–	$Pi = 1$
computations	$TN_c = \{\ \}$	–	$TN_c = \{\}$
	$TP_c = \{1\}$	–	$TP_c = \{1,3\}$
	$lb = 1/3$	–	$lb = 2/3$
	$ub = 2/3$	–	$ub = 2/3$
	$pc_1[\{A,B,D\}] = 1$	–	$pc_3[\{A,B\}] = 1$
		–	$pc_3[\{A,B,D\}] = 1$
Data structures		–	

consequently lower and upper bounds on the time prevalence are 0 and 1 respectively. However, in spatial dataset S_3^p, the candidate $\{A,B,C\}$ is not a subset of any local candidate and thus it cannot be spatially prevalent at the corresponding time moment. Consequently, $TN_e = \{3\}$ and upper bound on the time prevalence is lowered to 2/3.

Table 7 Processing of the candidate pattern {B,C,D}

Pattern	$\{B, C, D\}$		
Spatial dataset	S_1^p	S_2^p	S_3^p
Estimating results	$TN_e = \{\ \}$	$TN_e = \{\ \}$	$TN_e = \{\ \}$
	$TP_e = \{\ \}$	$TP_e = \{\ \}$	$TP_e = \{\ \}$
	$lb = 0$	$lb = 0$	$lb = 0$
	$ub = 1$	$ub = 1$	$ub = 1$
Accurate computations	$Pi = 1$	$Pi = 0.5$	–
	$TN_c = \{\}$	$TN_c = \{\}$	–
	$TP_c = \{1\}$	$TP_c = \{1, 2\}$	–
	$lb = 1/3$	$lb = 2/3$	–
	$ub = 1$	$ub = 1$	–
	$pc_1[\{B, C\}] = 1$	$pc_2[\{B, C\}] = 0.5$	–
	$pc_1[\{B, C, D\}] = 1$	$pc_2[\{B, C, D\}] = 0.5$	–
Data structures			–

Next, the accurate computations take place. The participation index computed for S_1^p is 1. Hence, TP_c set is updated to store time moment 1 and the lower bound on the time prevalence is updated to 1/3. As a side effect of the participation index computation algorithm (see Section 4.6), pc_1 cache is updated to store participation indices for patterns $\{A, B\}$ and $\{A, B, C\}$ (both equal to 1). Next, accurate computations for S_2^p are performed. This time, the participation index is equal to 0, which means that at the time moment 2, the candidate $\{A, B, C\}$ is not spatially prevalent. The TN_c set is updated accordingly as well as lower and upper bounds. Since the upper bound is now equal to 1/3 (which is less than $mintprev = 2/3$), we know that the candidate is not time prevalent. As a side effect of the participation index computation algorithm, pc_2 cache is updated with the participation index of $\{A, B\}$. Moreover, the pattern $\{A, B, C\}$ is stored in the npc_2 cache.

Since $\{A, B, C\}$ candidate is not time prevalent, the N set is updated with all of the candidate's subsets (one item smaller). Thus, $N = \{\{A, B\}, \{A, C\}, \{B, C\}\}$.

The candidate $\{A, B, D\}$ is processed next. All the computations regarding this candidate are presented in Table 6. Based on the contents of the caches and the sets of local candidates we are able to determine that the candidate cannot be spatially prevalent at S_2^p. Consequently, $TN_e = \{2\}$ and upper bound on time prevalence is $ub = 2/3$. Accurate computations are therefore needed only for S_1^p and S_3^p. In both cases, the computed participation index is equal to 1. Hence, the lower bound on time prevalence is first updated to 1/3 and next to 2/3. Since lower bound is greater or equal to $mintprev = 2/3$, we know that this candidate is time prevalent. As a side effect of the participation index computation algorithm pc_1 cache is updated with the participation index of $\{A, B, D\}$. Note that the participation index of the prefix $\{A, B\}$ was computed in previous iterations. Moreover, pc_3 cache is updated with participation indices of $\{A, B\}$ and $\{A, B, D\}$ at S_3^p.

Finally, the candidate $\{B, C, D\}$ is processed. All the computations regarding this candidate are presented in Table 7. Analysis of caches and local candidates does not yield any usable information. Hence, no results could be estimated and the lower and upper bound at this step are respectively 0 and 1. Accurate computations for S_1^p yield the participation index equal to 1, which raises the lower bound to 1/3. Moreover, pc_1 cache is updated with participation indices of $\{B, C\}$ and $\{B, C, D\}$. Accurate computations for S_2^p yield participation index equal to 0.5, which is still greater or equal to $minprev = 0.5$. Thus, the candidate is spatially prevalent at S_2^p and consequently, the lower bound on the time prevalence can be raised to 2/3. Since lower bound is greater or equal to $mintprev = 2/3$, we know that this candidate is time prevalent and no subsequent computations are necessary. Moreover, the pc_2 cache is updated with participation indices of $\{B, C\}$ and $\{B, C, D\}$.

Since all size 3 candidates are now processed, we need to determine the set of the next candidates. We remove from the set N all subsets of $\{A, B, D\}$ and $\{B, C, D\}$. Only the set $\{A, C\}$ remains, which in the absence of other candidates of this size in the $K(ST)$, is the only size 2 candidate.

Algorithm 5 Mining maximal MDCOPs

```
Require:
  • a set of global candidates for maximal MDCOPs $K(ST)$
  • sets of local candidates for maximal MDCOPs $K(S^p_t)$
  • $minprev$ and $mintprev$ thresholds
Ensure: a set of maximal MDCOPs $\mathbb{C}^T(ST)$
1: $\mathbb{C}^T(ST) \leftarrow \{\}$
2: $\forall_{S^p_t \in ST} lpc_t = \{\}$; $\forall_{S^p_t \in ST} npc_t = \{\}$; $\forall_{S^p_t \in ST} pc_t = []$;
3: $Y \leftarrow K(ST)$
4: while $|Y| > 0$ do
5:     $ml \leftarrow max\{|X| : X \in Y\}$
6:     if $ml = 2$ then
7:         $\mathbb{C}^T(ST) \leftarrow \mathbb{C}^T(ST) \cup (Y \cap C_2(ST))$
8:         break
9:     else
10:        $M \leftarrow \{X : X \in Y \wedge |X| = ml\}$, $Y \leftarrow Y \setminus M$
11:        $N \leftarrow \{\}$
12:        for $P \in M$ do
13:            $TP_e \leftarrow \{\}$, $TN_e \leftarrow \{\}$, $TP_c \leftarrow \{\}$, $TN_c \leftarrow \{\}$
14:            for $S^p_t \in ST$ do
15:                if $\nexists X \in K(S^p_t) : P \subseteq X$ then $TN_e \leftarrow TN_e \cup \{t\}$
16:                else if $\exists X \in npc_t : X \subseteq P$ then $TN_e \leftarrow TN_e \cup \{t\}$
17:                else if $\exists X \in lpc_t : P \subset X$ then $TP_e \leftarrow TP_e \cup \{t\}$
18:                if $lb(P) \geq mintprev$ or $ub(P) < mintprev$ then break
19:            end for
20:            if $lb(P) < mintprev$ and $ub(P) \geq mintprev$ then
21:                for $S^p_t \in ST : t \notin (TP_e \cup TN_e)$ do
22:                    $prev \leftarrow compute\_pi(S^p_t, P, pc_t, npc_t, minprev)$    ▷ Alg. 7
23:                    if $prev \geq minprev$ then $TP_c \leftarrow TP_c \cup \{t\}$
24:                    else $TN_c \leftarrow TN_c \cup \{t\}$
25:                    if $lb(P) \geq mintprev$ or $ub(P) < mintprev$ then break
26:                end for
27:            end if
28:            if $lb(P) \geq mintprev$ then
29:                $\mathbb{C}^T(ST) \leftarrow \mathbb{C}^T(ST) \cup \{P\}$
30:            else
31:                $N \leftarrow N \cup \{P' \subset P : |P'| = ml - 1\}$
32:                if $ml > 3$ then: for $t \in TP_c$ do: $lpc_t \leftarrow lpc_t \cup \{P\}$
33:            end if
34:        end for
35:        for $P \in N$ do: if $\nexists X \in \mathbb{C}^T(ST) : P \subset X$ then $Y \leftarrow Y \cup \{P\}$
36:    end if
37: end while
```

Candidates of this size can be easily verified, since their time prevalences were computed in Step 2. In Table 2 we can see that the candidate $\{A, C\}$ is time prevalent and thus it should be included in the result set.

Finally, we obtain three time prevalent maximal MDCOPs: $\{A, B, D\}$, $\{B, C, D\}$ and $\{A, C\}$.

4.6 Step 5a. Computing spatial prevalence

As we have mentioned before, the participation index computations are the most time consuming part of the algorithm. In this section, we provide an efficient algorithm which is used in Step 5 described in the previous section. The new algorithm is inspired by the solutions presented by Yao et al. (2016). In the approach presented in that paper, authors compress co-location instances of a single co-location pattern by inserting them into a trie structure (Fredkin, 1960). In our approach, we extend this solution by introducing a novel method for storing such tries. The new method not only further improves the compression ratio of co-location instances, but also allows to improve the performance of co-locations participation index computations and reuse partial results when finding instances of another co-location with a common prefix. Hence, this section can be logically divided into two parts: description of the data structures and description of the participation index computation algorithm. These parts are presented below as two subsections.

4.6.1 Data structures

A trie is a well known data structure, initially introduced for indexing and compressing of strings (Fredkin, 1960). Nevertheless, any sequence can be stored in it. The trie is a tree in which a sequence is represented as a path from the root to a leaf. If multiple sequences are inserted, common sequence prefixes are compressed, since they share the same paths.

In order to insert either a co-location pattern or a co-location pattern instance into a trie, we must represent it as a sequence. To convert a set to a sequence, all the items in it must be sorted. Hence, a total order on the set of features F is needed. Since there are no other requirements, such order is easy to define. Thus, both co-location patterns and co-location pattern instances can be stored in a trie. Because construction of tries is beyond the scope of this paper, we will not cover this subject. We refer the reader to (Fredkin, 1960; Yao et al., 2016) for details.

In the following paragraphs we gradually introduce the data structures used in our algorithm.

In Fig. 2, we present an exemplary trie for a set of instances of a co-location pattern $\{A, B, C, D\}$. In our approach we store levels of a trie as arrays. Each entry in such an array is a triple $(i^f, pos, visited)$, where i^f is a feature instance, pos is an index of a parent entry in a higher level array and *visited* is a flag which allows to optimize prevalence computations. Such arrays are called the *level arrays*.

Notice, that aside from instances of a single co-location, the same trie can be used to store instances of the co-location's prefixes. For example, a trie that stores instances of co-location $\{A, B, C, D\}$, can also store instances of $\{A, B, C\}$ and $\{A, B\}$. This is due to the fact that prefixes of co-location instances are instances of the corresponding co-location prefix. Thus, in order to retrieve all instances of some co-location prefix, one can follow paths from the nodes at level corresponding to the prefix's size towards the root of the trie. Based on these observations, we store instances of a co-location and all of its prefixes in a single trie. Figure 3 shows an exemplary trie which compresses instances of co-location $\{A, B, C, D\}$ and all of its prefixes. We call such tries the *instance tries*.

The presented compression method can be improved even further. Consider two co-locations with a common prefix, e.g. $\{A, B, C, D\}$ and $\{A, B, E\}$. A very important observation is that the instance tries for these two co-locations will have the same first n levels, where n is the size of the common prefix. For the two exemplary co-locations, two first levels of the instance tries (without the root) will be the same. Thus, when finding instances of a co-location, we can reuse several first levels of a trie corresponding to another co-location with a common prefix. For example, when building an instance trie for co-location $\{A, B, E\}$, we can reuse two first levels of a trie for co-location $\{A, B, C, D\}$ assuming it has been built earlier. Moreover, since these levels are the same, we do not need to store them multiple times in memory.

In order to take advantage of the last observation, we need to create a more complex data structure. We propose a two part structure. This first part is a collection of level arrays. It can be anything that allows for random access, e.g. another array. By storing the level arrays in a single collection, we lose information about which level arrays correspond to which tries. This information is stored in the second part of the data structure. The second part of the structure is also a trie, however it stores co-location patterns instead of co-location pattern instances. Such a trie is called a *co-location trie*. In this trie, a single node

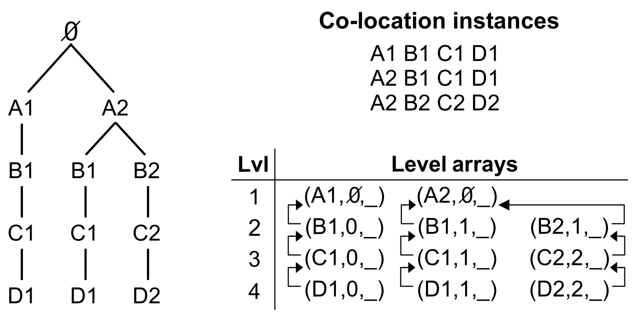


Fig. 2 A trie with co-location instances

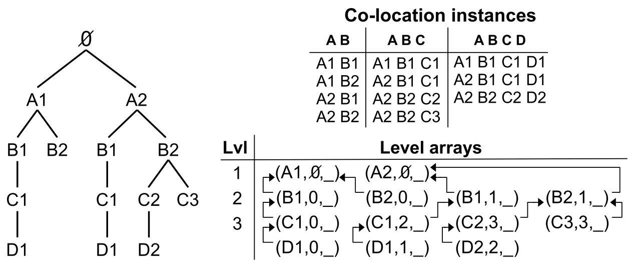


Fig. 3 A trie with instances of a co-location and its prefixes

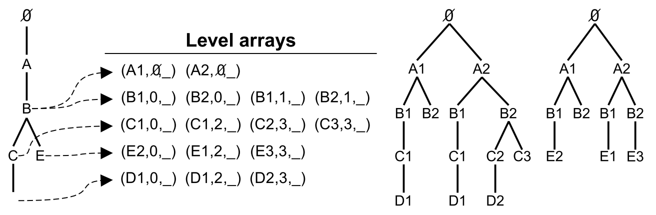
Algorithm 6 Building a co-location trie

```
Require:
  • a spatial dataset $S^p_t$ with iCPI-tree, a co-location trie and a relation $R$
  • a co-location $P$
Ensure: a co-location trie and a collection of level arrays are updated for $S^p_t$
 1: $V \leftarrow$ a sorted array with features from $P$
 2: $d \leftarrow$ the length of $P$ prefix already stored in a co-location trie
 3: if $d = |P|$ then abort, all level arrays are computed
 4: if $d < 2$ then
 5:     $L0 \leftarrow$ empty level array, $L1 \leftarrow$ empty level array
 6:     for every $i^{V[0]}$ from first two levels of $S^p_t$ iCPI-tree do
 7:         if $N(i^{V[0]}, V[1], S^p, R) = \emptyset$ then continue
 8:         append $(i^{V[0]}, \emptyset, false)$ to $L0$
 9:         $w \leftarrow |L[0]| - 1$
10:         for $i^{V[1]} \in N(i^{V[0]}, V[1], S^p_t, R)$ do append $(i^{V[1]}, w, false)$ to $L1$
11:     end for
12:     Store size 2 $P$ prefix in co-loc. trie and add ref. to $L0$ and $L1$ to the leaf
13:     $d \leftarrow 2$
14: end if
15: $L \leftarrow$ an array of references to level arrays retrieved for size $d$ prefix of $P$
16: for $j \in d, \ldots, |P| - 1$ do
17:     Allocate a new level array and store a reference to it at $L[j]$
18:     for $parent\_pos \in 0, \ldots, |L[j-1]|$ do
19:         $CO \leftarrow \bigcap_{i^f \in \text{path to the root starting at } L[j-1][parent\_pos]} N(i^f, V[j], S^p_t, R)$
20:         Append to $L[j]$ entries $(i^f, parent\_pos, false)$ where $i^f \in CO$
21:     end for
22:     Add node to co-loc. trie for size $j$ $P$ prefix and copy reference from $L[j]$
23: end for
```

corresponds to a single co-location pattern, which is composed of features on a path from the node to the root. Each node stores a reference to a level array in the collection. Thus, to reconstruct an instance trie corresponding to some co-location, one needs to retrieve all references to level arrays on the path from the co-location trie's node to the root. Due to the specificity of co-location patterns, the above idea needs a slight tweak to make it work correctly. Note that the first level of the instance trie corresponds to single feature instances. Hence, every level array corresponding to the first level of the trie would need to contain all feature instances of a single feature. To avoid this, we propose to create first and second level of the instance trie together and store only such instance pairs that are neighbors. In other words, we do not store size 1 co-location pattern instances. As will be apparent later, such instances are not needed anyway. As a consequence to this change, the nodes at the first level of the co-location trie should not store any references. On the other hand, nodes at the second level should store references for the first and the second level of the corresponding instance trie. Figure 4 shows two instance tries for a set of co-location instances, as well as their representation in the form of a co-location trie and a corresponding collection of level arrays.

4.6.2 Participation index computations

Having described the data structures, we now continue to the participation index computation algorithm. The algorithm


Ø
A
B
C
E
Level arrays
(A1,Ø,_) (A2,Ø,_)
(B1,0,_) (B2,0,_) (B1,1,_) (B2,1,_)
(C1,0,_) (C1,2,_) (C2,3,_) (C3,3,_)
(E2,0,_) (E1,2,_) (E3,3,_)
(D1,0,_) (D1,2,_) (D2,3,_)
Ø
A1 A2
B1 B2 B1 B2
C1 C1 C2 C3
D1 D1 D2
Ø
A1 A2
B1 B2 B1 B2
E2 E1 E3

Co-location instances

A B	A B C	A B E	A B C D
A1 B1	A1 B1 C1	A1 B1 E2	A1 B1 C1 D1
A1 B2	A2 B1 C1	A2 B1 E1	A2 B1 C1 D1
A2 B1	A2 B2 C2	A2 B2 E3	A2 B2 C2 D2
A2 B2	A2 B2 C3		

Fig. 4 A co-location tree with the corresponding collection of level arrays

Algorithm 7 Computing the spatial prevalence (the participation index).

```
Require:
    • a spatial dataset $S_t^p$ with iCPI-tree, a co-location trie and a relation $R$
    • a candidate set of features $P$
    • a prevalence cache $pc_t$ for spatial dataset $S_t^p$
    • non prevalent patterns cache $npc_t$ for spatial dataset $S_t^p$
    • a minimal spatial prevalence threshold $minprev$
Ensure:
    • a spatial prevalence $prev$ of the candidate $P$
    • a prevalence cache $pc_t$ for spatial dataset $S_t^p$ is updated
    • non prevalent patterns cache $npc_t$ for spatial dataset $S_t^p$ is updated
1: if $P$ in $pc_t$ then $prev \leftarrow pc_t[P]$; return
2: Identify instances of $P$ if not available in the co-location trie    ▷ Alg. 6
3: $L \leftarrow$ an array of references to lev. arr. retrieved for $P$ from the co-loc. trie
4: Set all $visited$ flags in all level arrays in $L$ to $false$
5: $U \leftarrow$ an array of empty sets of size $|P|$
6: $nonprev\_P \leftarrow null$
7: for $lev = |P| - 1, \ldots, 1$ do
8:     $P' \leftarrow$ size $lev + 1$ prefix of $P$    ▷ In 1st iteration P'=P
9:     if $P'$ in $pc_t$ then break
10:    for $(i^f, pos, visited) \in L[lev] : visited = false$ do
11:        $U[lev] \leftarrow U[lev] \cup \{i^f\}$
12:        $parent \leftarrow pos$
13:        for $k \in lev - 1, \ldots, 0$ do
14:            $(i^g, new\_parent, parent\_visited) \leftarrow L[k][parent]$
15:            if $parent\_visited = true$ then break
16:            $U[k] \leftarrow U[k] \cup \{i^g\}$
17:            $L[k][parent] \leftarrow (i^g, new\_parent, true)$
18:            $parent \leftarrow new\_parent$
19:        end for
20:    end for
21:    $pc_t[P'] \leftarrow min\{|U[k]|/count(P'[k], S_t^p) : k \in 0, \ldots, lev\}$
22:    if $pc_t[P'] < minprev$ then $nonprev\_P \leftarrow P'$
23: end for
24: if $nonprev\_P <> null$ then
25:    Remove all supersets of $nonprev\_P$ from $npc_t$
26:    $npc_t \leftarrow npc_t \cup \{nonprev\_P\}$
27: end if
28: $prev \leftarrow pc_t[P]$
```

itself can be divided into two parts. The first part is used to construct an instance trie of a candidate and store it in the level array collection as well as the co-location trie. If possible, previously computed level arrays should be reused. The first part is presented in Algorithm 6. The second part is executed once the complete instance trie is known. This part computes the actual participation index value. It is presented in Algorithm 7.

Let us start with the first part (Algorithm 6). The first task performed in this algorithm is sorting the candidate *P*, so that it becomes a sequence. Next, we determine how many levels of a candidate's instance trie were built previously (line 2). If all levels are available, subsequent computations are aborted (line 3). If not all levels have been built, there are two other possibilities. There are either no available results to reuse or there exists a partially built instance trie (there are at least two first levels available, i.e. at least two first features of *P* are in the co-location trie).

In the first case, we need to build two first levels of the trie (lines 5–13) and then proceed to the second case. In the second case, we build all the remaining levels (lines 15–23).

The first two levels can be easily constructed based on the data stored in the iCPI-tree structure. First, new level arrays are allocated (line 5). Next, we scan two first levels of the iCPI-tree corresponding to the current spatial dataset to retrieve instances of a feature *V*[0] (the first feature in an ordered candidate *P*) with neighbors (lines 6–11). For each such instance, we determine whether it has neighbors with a feature $V[1]$ (line 7). If such neighbors exist, then appropriate entries are added to the new level arrays: $L0$ and $L1$ (lines 8–10). Finally, the co-location trie is updated with nodes representing size 2 prefix of the candidate. The node corresponding to the feature $V[1]$ is associated with references to the level arrays $L0$ and $L1$.

Construction of the remaining levels starts with the retrieval of the references to the levels that have already been built (line 15). Subsequent levels are constructed one by one in a loop (lines 16–23). A new level array is constructed as follows. We start by scanning the entries in the level array of the previous (i.e. parent) level (lines 18–21). Each of these entries corresponds to an instance of the candidate's prefix with the appropriate length. In order to build a next level, we must find common neighbors with the feature $V[j]$ of all feature instances in the prefix instance (i.e. all the feature instances on the path to the root), where j is the number of the new level. Neighbors with feature $V[j]$ can be easily retrieved from iCPI-tree. Thus, to find such common neighbors, we just need to intersect neighbor sets retrieved from the iCPI-tree for each of the feature instances on the path to the root (line 19). Each such common neighbor is used to create a new entry in the new level array (line 20). After the new level array is finished, a new node is appended to the co-location trie along with the reference to the newly created level array (line 22). The algorithm ends once all the level arrays are inserted into the co-location trie. Please note that the above method is an adaptation of the co-location instance identification algorithm presented by Wang et al. (2009).

We shall now describe Algorithm 7, which is used for computing participation indices of candidates. As will be shown later, the algorithm not only computes the participation index of a candidate, but also of all of its prefixes.

In order to compute participation index of a candidate co-location P, it is necessary to find unique feature instances appearing in the candidate's instances. Therefore, the first step of this algorithm is the execution of Algorithm 6 to ensure that all candidate's instances have been identified (line 2). Next, the corresponding instance-trie is retrieved from the co-location trie in the form of an array L of references to consecutive level arrays (line 3). The *visited* flags in the retrieved level arrays are reset to *false* since they might have been modified by some previous computations (line 4). To obtain unique feature instances mentioned earlier, we need a data structure for storing them and removing the duplicates. For this purpose we allocate an array of sets U, one set for every feature of P (line 5).

For the sake of clarity, we will now skip some parts of the code to explain the main idea behind the algorithm. Let us assume that the outer for loop (lines 7 and 23) does not exist, and the *lev* variable is constant and equal to $|P| - 1$. Note, that in such a case $P' = P$ and we compute only the participation index of the candidate P.

We retrieve feature instances from the instance trie by traversing it in a specific way. We start at each leaf (entry at the lowest level of the instance trie - level number *lev*) (lines 10–20) and we travel towards the root (lines 13–19). We store all feature instances on the path in the corresponding sets in the U array (lines 11 and 16). Note that since this is a tree, it might happen that the same node can be reached from multiple lower level entries. To cope with this problem, we use the aforementioned flag *visited*, to mark visited nodes (line 17). Note that if a node has already been visited, then all of its parents must have been visited as well. Thus, when a visited node is reached, the travel towards the root can be aborted (line 15). After all paths have been retrieved, and the corresponding feature instances stored in the U sets, the participation index is computed (line 21). The value of the participation index is stored in the pc_t cache (described in the previous section) and returned as a result at the end of the algorithm (line 28).

Let us notice that most of the work needed to compute a one item smaller prefix P' of the candidate P has already been done. The U sets contain most of the feature instances of the P' co-location instances. What is left, is to traverse unvisited paths that start at level $lev - 1$, update the U sets and recompute the participation index. Now let us return to the outer for loop (line 7). In each iteration, the *lev* variable is decreased by one. Moreover, in the middle for loop, we iterate over unvisited nodes in level array number *lev*. Therefore, the next iteration of the outer for loop will perform computations that are necessary to compute the participation index of the prefix P'. The same line of reasoning can be applied to any size of the candidate's prefix. Thus, the outer for loop iterates over consecutive *lev* values up to one (which corresponds to size 2 prefixes) and computes participation indices for all prefixes.

Let us now consider line 9. In this line, we check whether the participation index of the current prefix P' has already been computed or not. If it has, then all of subsequent prefixes are associated with their participation indices as well and we can safely abort subsequent computations.

The final part of this algorithm is the construction of the npc_t cache. Before the actual participation index computations take place, we initialize a variable *nonprev_P* (line 6). This variable will store the smallest, spatially non prevalent candidate prefix found during the computations. We compare the computed participation index with the *minprev* threshold and if the prefix is not spatially prevalent, it is assigned to the *nonprev_P* variable (line 22). In the end of the algorithm, we verify whether a spatially non prevalent prefix was found (line 24) and if it was, the npc_t cache is updated. First, all supersets of *nonprev_P* co-location are removed (line 5). Second, the *nonprev_P* co-location is added to the npc_t cache (line 26).

Example 6 In the previous example three maximal MDCOPs were found. All the results computed during execution of the mining algorithm are presented in Tables 5, 6 and 7.

Let us now consider the spatial dataset S_1^p. When the participation index is computed for candidate $\{A, B, C\}$ no structures are constructed yet. Hence, the whole co-location trie as well as instance trie are built. These structures are shown in Table 5. Due to space limitations we only present graphical representation of instance tries (not the level array representation). Nevertheless remember that every level of an instance trie is represented as a separate array. When the participation index of the candidate $\{A, B, D\}$ is computed, most of the instance trie is already built (Table 6). We can reuse level arrays for the feature A and B and only add level array for the feature D. New data is drawn as black, while old data is drawn as gray. Finally, when the participation index of the $\{B, C, D\}$ candidate is computed, no data can be reused. Hence completely new paths are added to the co-location trie and instance trie (Table 7) and consequently three new level arrays for B, C and D features are created.

Let us now consider the spatial dataset S_2^p. The participation index is computed for the candidates $\{A, B, C\}$ and $\{B, C, D\}$. In the case of the first of the two patterns we can notice that the level array for the feature C is empty. Hence, participation index of the candidate in S_2^p is zero. However, level arrays for $\{A, B\}$ prefix were built as well and its participation index was also computed and stored in the cache pc_2. The computation of the participation index of $\{B, C, D\}$, similarly to S_1^p, requires completely new paths (see Table 7).

The participation index in the spatial dataset S_3^p was computed only once, for $\{A, B, D\}$ candidate. All the found structures are shown in Table 6. For the sake of the example, let us analyse how the participation index computations would work. First, the level array representing the level corresponding to the D feature is scanned. First entry is $D1$. Hence, this object is added to the appropriate set. It's parent entry – $B1$, and parent entry of $B1$ – $A1$, are also added to the appropriate sets. All the nodes on the path to the root are marked as visited. Similar operations are performed for the next entry in the level array corresponding to the D feature. Finally, the last entry in this array (the second $D2$) is analysed. $D2$ is added once again to the appropriate set as well as its parent $B2$. Note however, that we do not continue to travel to the node $A2$ since it was already visited. Therefore, the sets should now contain unique items for every feature and the participation index of $\{A, B, D\}$ can now be computed. We can now reuse the computations to find participation index of the prefix $\{A, B\}$. We now scan the level array for the feature B in order to find all the unmarked entries. In this case there are none. We can therefore use the sets for features A and B to compute the participation index of $\{A, B\}$. In case there were such unmarked entries, we would just have to insert the corresponding objects and all the yet unvisited parents into the appropriate sets. The sizes of the updated sets can be used to compute the new participation index value.

5 Experiments

Algorithms and Testing Environment Since, up to now, no solutions for mining maximal MDCOPs were proposed in the literature, we compare our algorithm with two "competitors":

- FastMDCOP-Miner - a state-of-the-art MDCOP mining algorithm. Unfortunately, this algorithm mines all MDCOPs, not just the maximal ones.
- A vanilla variant of MAXMDCOP-Miner stripped of many optimizations. We treat this variant as an initial viable solution. We codename this variant MAXMDCOP-Miner STD, while the full version is called MAXMDCOP-Miner ENH.

The following features were stripped from the MAXMDCOP-Miner STD:

- No upper and lower bounds on time prevalence are computed in Step 5. Instead, all datasets in ST are always analyzed.
- The algorithm uses only the accurate participation index computations. Hence, the sets TN_e, TP_e, TN_c and TP_c are not computed and consequently caches *npc* and *lpc* are not used (and are not created). Instead we only count the number of times a candidate is spatially prevalent if order to compute the time the prevalence.
- The participation index computation algorithm computes only the participation index of a single candidate (without prefixes). The instance tree is always created from scratch and removed after the participation index is computed. Hence, the cache *pc* is not used (and is not created).

All implementations are in Python3 language. Experiments have been conducted on a machine with Intel Xeon Gold 6138 2.00GHz CPU and 24GB RAM.

Datasets The procedure to generate synthetic data was based on the method used by Celik et al. (2006). The overall idea was to prepare persistent and transient patterns, generate their instances and instances of noise. Persistent patterns occur in at least specified number of time moments, while transient patterns occur in a less than specified number of time moments. At the beginning, a set of PAT_{ct} initial patterns was created. The initial patterns represent subsets of feature types that could be involved in MDCOPs. For each initial pattern, its size was randomly chosen using the Poisson distribution with mean PAT_{avs}. Next, initial features from a user defined set of F_{ct} were assigned randomly to the initial patterns. Having the set of initial patterns, persistent and transient pattern sets were created by dividing the initial patterns in the proportion described by RAT_p parameter - a ratio of persistent patterns over transient patterns. Instances of patterns were placed in a spatial framework defined as a square of side *dim*. For simplicity, the spatial framework was divided into square cells of size equal to the neighborhood distance *dist*. Each pattern instance was placed in a square randomly chosen from the spatial framework. For each pattern, its instances were placed in a randomly chosen number of time moments from the set of TF_{ct} frames w.r.t. *mintprev* parameter, i.e. persistent patterns had to occur in at least *mintprev* fraction of time moments. Notice that *mintprev* also defines the maximal number of time moments in which a transient pattern instance can occur. The number of instances of the particular pattern in a given time moment was randomly chosen from the Poisson distribution with mean INS_{avc}. Finally, a noise was generated by placing $NOISE_{ct}$ objects in the spatial framework. Each noise object was assigned a spatial feature randomly chosen from a set of noise features. This set consisted of RAT_n percent of F_{ct} initial features. Notice that some of those features could also participate in generated patterns.

Using the aforementioned procedure, we have prepared two datasets, namely SD1 and SD2 with 136K objects and 29K objects respectively. To generate those datasets we have used the parameter values presented in Table 8. In both cases the spatial framework size and neighboring distance were the same ($dim = 10000$ and $dist = 10$). Dataset SD1 contains many small MDCOP patterns, whereas SD2 has fewer patterns, although their size can be significantly larger. The total number of spatial features used to generate SD1 is twice as small as in the second dataset. Considering the greater ratio for noise features and the number of instances of such features, SD1 contains much shorter patterns than SD2.

The real world dataset RD contains the positions of pigeons from the animal study (Zannoni et al., 2020). Our analysis was limited to the data from a single day and we used linear interpolation to calculate pigeons' positions for each of 1440 time moments (one per minute). There were 29192 objects and 29 spatial features in the RD dataset.

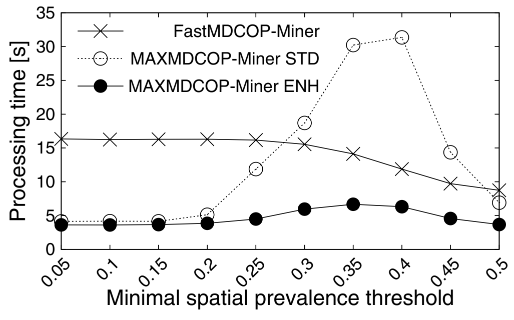
Results In the experiments, the processing time of the algorithms and their memory usage were studied. By memory usage, we mean the largest memory requirement during the execution of the algorithm. Measurements were made for varying the values of three parameters: the minimal spatial prevalence threshold (*minprev*), the minimal time prevalence threshold (*mintprev*) and the maximal distance threshold (*maxdist*). In a given experiment, only one of the parameters was changed, while the other two were set to particular values. We will begin the discussion of the results by analyzing the processing times of the algorithms.

In the first experiment, we observed the impact of changing the minimal prevalence threshold on the processing time of the synthetic datasets SD1 and SD2. In this experiment, the minimal time prevalence threshold was set at 0.3 and the maximal distance threshold was set at 10. The results are shown in Fig. 5a and b for SD1 and SD2 respectively. In general, the lower the *minprev*, the greater the probability of finding a pattern in a given frame. Thus, processing times should decrease with decreasing values of *minprev*. This is exactly the behavior that can be observed for the FastMDCOP-Miner algorithm. In both versions of the proposed algorithm, a lower *minprev* will have a higher likelihood of producing long and prevalent patterns, eliminating the need to check their subsets. For small values of *minprev*, MAXMDCOP-Miner in both versions is nearly 4 times faster than FastMDCOP-Miner. For larger values of *minprev*, the performance difference between the two versions of MAXMDCOP-Miner becomes clear. The STD version, due to lack of optimizations, generates significantly more computations, resulting in processing times that exceed the processing times achieved by ENH version as well as FastMDCOP-Miner. The ENH version, for both datasets, is more efficient than FastMDCOP-Miner over the entire range of examined *minprev* values.

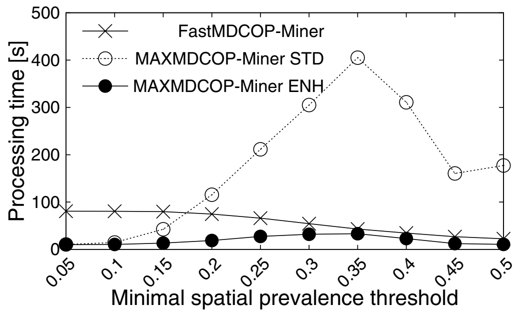
In the second experiment, we used the same SD1 and SD2 data sets to explore the effects of changing *mintprev* on processing time. The minimal prevalence threshold was set at 0.3 and the maximum distance threshold was set once again at 10. Figure 5c and d presents the achieved results. For the FastMDCOP-Miner algorithm, increasing *mintprev* has almost a linear effect on reducing the processing time. This is, to some extent, a result of the way the synthetic data is generated. For slightly more diverse data, the relationship does not need to be linear. MAXMDCOP-Miner ENH was once again the fastest method and achieved the best results for low and high values of *mintprev*. For low *mintprev*, there is a higher chance that

Table 8 Synthetic data generator parameters

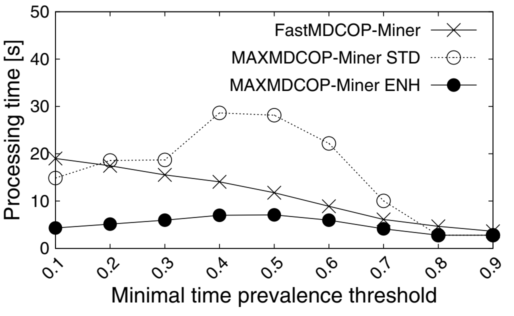
Dataset	PAT_{ct}	PAT_{avs}	F_{ct}	RAT_p	TF_{ct}	*mintprev*	INS_{avc}	$NOISE_{ct}$	RAT_n
SD1	15	5	100	0.4	100	0.6	25	20000	0.4
SD2	5	10	200	0.3	50	0.6	15	10000	0.2



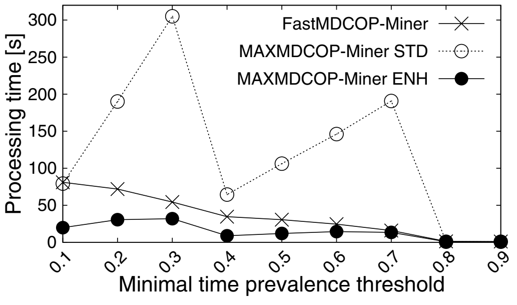
(a) dataset SD1



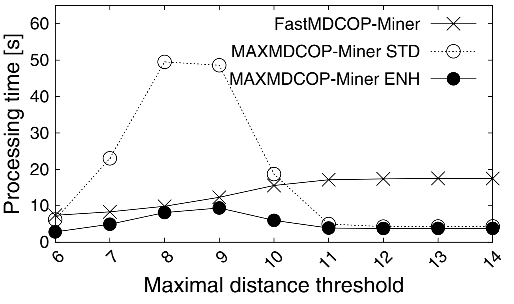
(b) dataset SD2



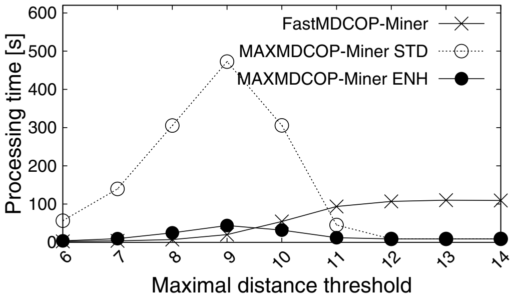
(c) dataset SD1



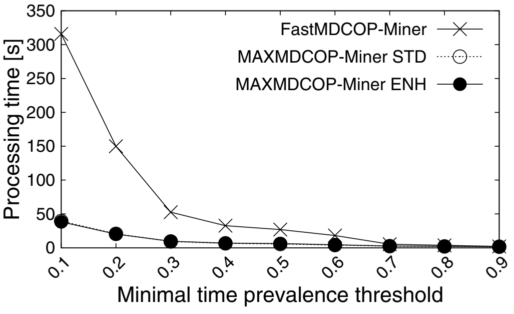
(d) dataset SD2



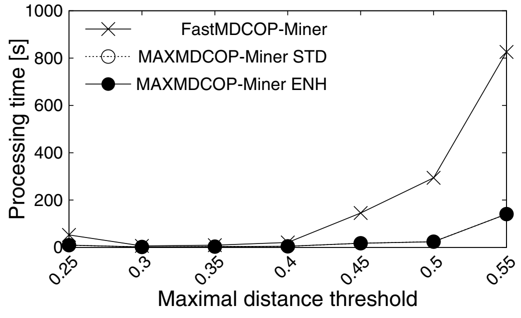
(e) dataset SD1



(f) dataset SD2



(g) dataset RD



(h) dataset RD

Fig. 5 Results of the experiments - processing times

lower or upper bound time prevalence filtering will take place, resulting in a reduced number of computations. The same cannot be said for the STD version, where mechanisms for more advanced filtering are not present. Hence, the performance of such a basic version of the algorithm is almost always not better than that of FastMDCOP-Miner. For high values of *mintprev*, the number of patterns decreases as well as the performance gap among all algorithms.

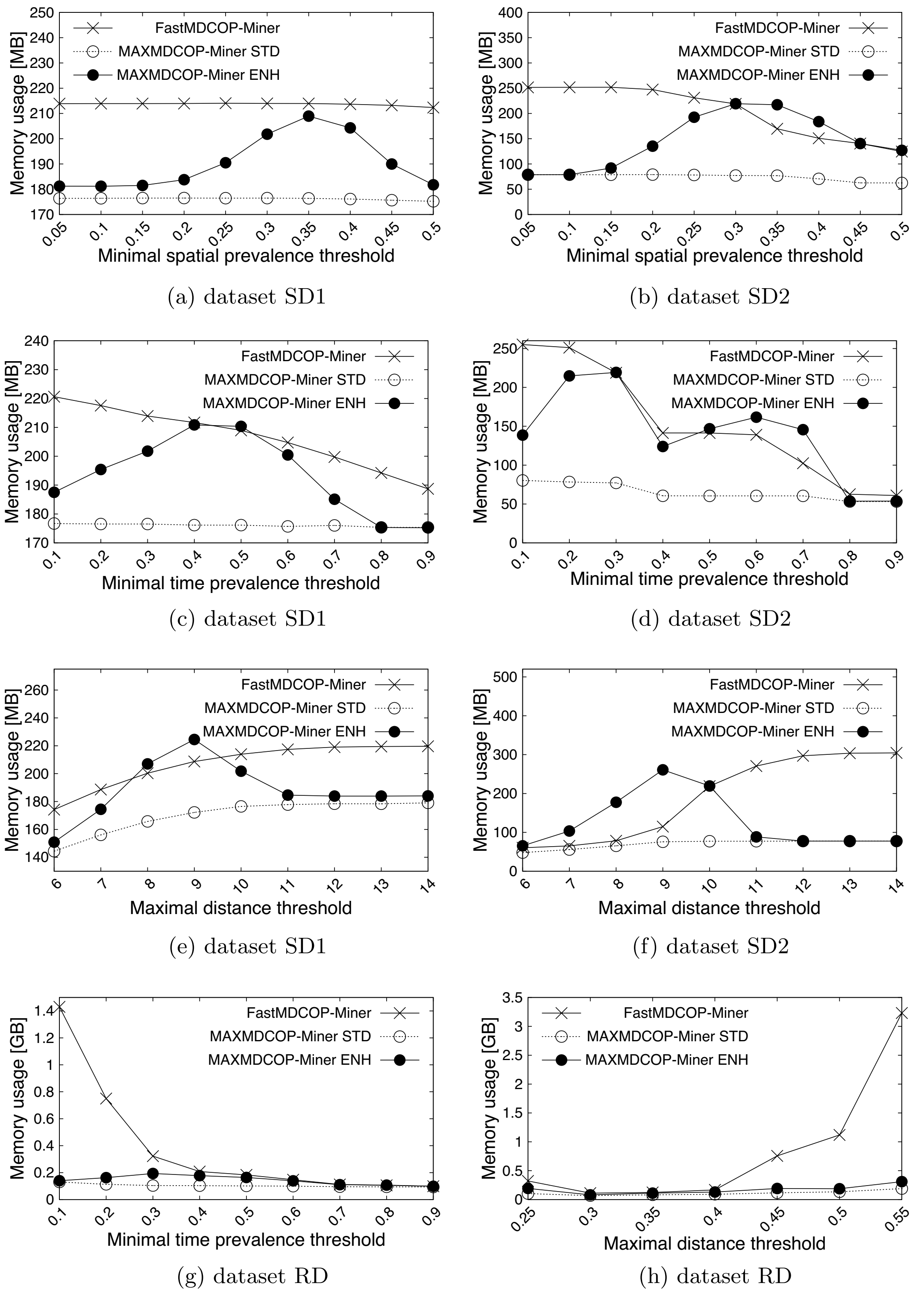


Fig. 6 Results of the experiments - memory usage

In the third experiment, we tested how the maximal distance *maxdist* affects the performance on synthetic datasets. In this experiment, both the minimal prevalence threshold and the minimal time prevalence threshold were set at 0.3. The results for datasets SD1 and SD2 are presented in Fig. 5e and f. In classical co-location pattern discovery, increasing *maxdist* significantly affects the number of candidates, instances, and thus the total processing time.

Similar relationships can be observed for the FastMDCOP-Miner algorithm. In particular, there is a clearly observable increase in processing times when *maxdist* reaches a value of 10. This is related to the parameters used for the synthetic data generator, namely *dist* set at 10. For high *maxdist* values, the number of instances (and candidates) can increase very rapidly and the Apriori-based approach is inefficient in comparison to the new method (both STD and ENH versions). For lower values of *maxdist*, the STD version generates many unnecessary candidates, resulting in the worst processing times. The ENH version is similar in execution times to FastMDCOP-Miner and is slightly faster or slightly slower depending on the dataset. It is worth noticing that for higher values of *maxdist*, the new method can be an order of magnitude faster than FastMDCOP-Miner.

The next experiment involved a study of the processing times of a real dataset. As with the synthetic data, we examined the impact *mintprev* and *maxdist* on processing time. Due to the limited number of observed pigeons, we skip the *minprev* threshold assuming that all candidates are spatially prevalent. Figure 5g presents the effect of varying *mintprev*, while *maxdist* was set at 0.4. Figure 5h shows the impact of changing *maxdist*, while *mintprev* was set at 0.3. These values were arbitrarily chosen to present the most interesting results. The collected data confirms the results achieved for the synthetic dataset. With respect to the synthetic data, one can observe that for real data the relationships are definitely nonlinear. Nevertheless, in all cases the new method (in both versions) was more efficient than the FastMDCOP-Miner.

In addition to examining the processing times of the algorithms, the level of demand for memory resources was also studied. The results showing the maximum memory usage are shown in Fig. 6a-h, which correspond to the time performance experiments shown in Fig. 5a-h. In all performed experiments, the STD version of the MAXMDCOP-Miner algorithm showed the least memory requirements. This is due to the fact that MAXMDCOP-Miner always utilizes data compression structures. At the same time, the STD version lacks elements related to data caching, which sometimes require quite significant memory resources. The ENH version, which uses (among others) various types of caches, for synthetic data in certain ranges of examined parameters variability, shows slightly higher demand for operating memory than FastMDCOP-Miner. However, for the most demanding parameter ranges (i.e. low *minprev* or *mintprev* or high *maxdist*), the memory requirements are significantly lower than for FastMDCOP-Miner. For the examined real world dataset, again the MAXMDCOP-Miner algorithm in the STD version was the least memory intensive; the MAXMDCOP-Miner in the ENH version performed only slightly worse. By far the largest memory requirements were for FastMDCOP-Miner.

6 Summary and Future Work

This is the first algorithm that tackles the problem of mining Maximal Mixed-Drove Co-occurrence Patterns. To achieve the best performance, we have resigned from the traditional Apriori generate-and-test approach and adapted the method based on maximal cliques mining that was successfully applied in mining of classical co-location patterns. We have developed novel data structures that provide efficient compression of instances of MDCOPs. We have also proposed a new participation index computation algorithm, which uses the aforementioned data structures to cache and reuse intermediate results of computations.

We have tested the proposed solution using both synthetic and real world datasets. Gathered results show that new approach is more efficient than the current state-of-the-art method when comparing processing times as well as memory requirements. The more demanding parameter values (i.e. low spatial or time prevalence thresholds or high maximum distance threshold), the greater the performance gain. Therefore, our method can be particularly useful in analyzing dense datasets that can contain patterns of significant length.

The important limitation of the proposed method is the requirement to fit all structures in memory, which for some, very large MDCOP mining problems can be impossible. Therefore, one of the main directions of further development is to design structures that can be efficiently stored and processed on disk drives.

There are also some more technical elements that can be improved in the MAXMDCOP-Miner, namely the maintenance of *visited* flags and data structures for sets operations. *Visited* flags are used to prevent visiting the same nodes in a tree when traversing from lower level entries. Although these flags prevent unnecessary operations, they must be periodically set to *false* in retrieved level arrays, which imposes an additional cost. The second mentioned improvement concerns the use of more specialized structures for performing operations on sets, which are widely used in the algorithm.

In the future work, we also intend to parallelize the MDCOP mining process, which can result in order of magnitude speedups. Currently, we are working on detailed calculations of the MAXMDCOP-Miner complexity.

Acknowledgements This research has been partially supported by the statutory funds of Poznan University of Technology.

Declarations

Conflicts of interest The authors have no competing interests to declare that are relevant to the content of this article.

References

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *Proc. of the 20th international conference on very large data bases* (pp. 487–499). Morgan Kaufmann Publishers Inc.,

Andrzejewski, W., & Boinski, P. (2021). Maximal mixed-drove co-occurrence patterns. In L. Bellatreche, M. Dumas, P. Karras, & R. Matulevičius (Eds.) *Advances in databases and information systems* (pp. 15–29.) Springer

Andrzejewski, W., & Boinski, P. (2018). Efficient spatial co-location pattern mining on multiple gpus. *Expert Systems with Applications, 93*(Supplement C), 465–483. https://doi.org/10.1016/j.eswa.2017.10.025.

Andrzejewski, W., & Boinski, P. (2019). Parallel approach to incremental co-location pattern mining. *Information Sciences, 496,* 485–505.

Cao, H., Mamoulis, N., & Cheung, D. W. (2006). Discovery of collocation episodes in spatiotemporal data. In *Proc. of the 6th international conference on data mining*. ICDM '06 (pp. 823–827). IEEE Computer Society, Washington, DC, USA

Celik, M., Shekhar, S., Rogers, J. P., & Shine, J. A. (2006). Sustained emerging spatio-temporal co-occurrence pattern mining: A summary of results. In *Proc. of the 18th IEEE international conference on tools with artificial intelligence* (ICTAI'06) (pp. 106–115)

Celik, M., Shekhar, S., Rogers, J. P., & Shine, J. A. (2008). Mixed-drove spatiotemporal co-occurrence pattern mining. *IEEE Transactions on Knowledge and Data Engineering, 20*(10), 1322–1335.

Fredkin, E. (1960). Trie memory. *Communications of the ACM, 3*(9), 490–499.

Hamdi, S. M., Aydin, B., & Angryk, R. A. (2016). A pattern growth-based approach for mining spatiotemporal co-occurrence patterns. In *Proc. of the 16th IEEE international conference on data mining workshops* (pp. 1125–1132)

Huang, Y., Zhang, L., & Zhang, P. (2008). A framework for mining sequential patterns from spatio-temporal event data sets. *IEEE Transactions on Knowledge and Data Engineering, 20*(4), 433–448.

Li, Z. (2014). *Spatiotemporal pattern mining: Algorithms and applications* (Vol. 9783319078212, pp. 283–306). New York: Springer, International Publishing. https://doi.org/10.1007/978-3-319-07821-2_12.

Pillai, K. G., Angryk, R. A., Banda, J. M., Schuh, M. A., & Wylie, T. (2012). Spatio-temporal co-occurrence pattern mining in data sets with evolving regions. In *2012 IEEE 12th international conference on data mining workshops* (pp. 805–812). IEEE

Qian, F., Yin, L., He, Q., & He, J. (2009). Mining spatio-temporal co-location patterns with weighted sliding window. In *2009 IEEE international conference on intelligent computing and intelligent systems* (Vol. 3 pp. 181–185). https://doi.org/10.1109/ICICISYS.2009.5358192

Qian, F., He, Q., & He, J. (2009). Mining spread patterns of spatio-temporal co-occurrences over zones. In O. Gervasi, D. Taniar, B. Murgante, A. Laganà, Y. Mun, & M. L. Gavrilova (Eds.), *Computational science and its applications - ICCSA 2009* (pp. 677–692). Springer.

Shekhar, S., & Huang, Y. (2001). Discovering spatial co-location patterns: A summary of results. In *Proc. of the 7th international symposium on spatial and temporal databases (SSTD 2001)*. Lecture Notes in Computer Science, (Vol. 2121 pp. 236–256). Springer

Tran, V., Wang, L., Chen, H., & Xiao, Q. (2021). Mcht: A maximal clique and hash table-based maximal prevalent co-location pattern mining algorithm. *Expert Systems with Applications, 175,* 114830.

Wang, L., Zhou, L., Lu, J., & Yip, J. (2009). An Order-clique-based approach for mining maximal co-locations. *Information Sciences, 179*(19), 3370–3382.

Wang, L., Bao, Y., & Lu, J. (2009). Efficient discovery of spatial Co-Location patterns using the iCPI-tree. *The Open Information Systems Journal, 3*(2), 69–80.

Wang, Z., Han, T., & Yu, H. (2019). Research of MDCOP mining based on time aggregated graph for large spatio-temproal data sets. *Computer Science and Information Systems, 16,* 32–32.

Wei, W. W. S. (2006). Time series analysis: Univariate and multivariate methods, 2nd edn. Pearson Addison Wesley

Yang, H., Parthasarathy, S., & Mehta, S. (2005). A generalized framework for mining spatio-temporal patterns in scientific data. In *Proc. of the 11th ACM SIGKDD international conference on knowledge discovery in data mining*. KDD '05, (pp. 716–721). ACM

Yao, X., Peng, L., Yang, L., & Chi, T. (2016). A fast space-saving algorithm for maximal Co-location pattern mining. *Expert Systems with Applications, 63*(C), 310–323.

Yoo, J. S., & Bow, M. (2011). Mining maximal Co-located event sets. In J.Z. Huang, L. Cao, & J. Srivastava (Eds.) *Proc. of the 15th pacific-asia conference on knowledge discovery and data mining (PAKDD 2011)*. Lecture Notes in Computer Science (Vol. 6634, pp. 351–362). Springer

Yoo, J. S., & Shekhar, S. (2006). A joinless approach for mining spatial colocation patterns. *IEEE Transactions on Knowledge and Data Engineering, 18*(10), 1323–1337. https://doi.org/10.1109/TKDE.2006.150.

Zannoni, N., Wikelski, M., Gagliardo, A., Raza, A., Kramer, S., Seghetti, C., et al. (2020). Identifying volatile organic compounds used for olfactory navigation by homing pigeons. *Scientific Reports UK, 10*(15879), 1–16.

Witold Andrzejewski is an assistant professor of Computer Science at Poznan University of Technology in Poznan, Poland. He received PhD from the same university in 2008. His main research interests include spatial data analysis, optimization of query processing of complex data structures and data mining via auxiliary structures (indices) as well as hardware acceleration and parallelization (GPU based) of some typical database and data mining operations. Witold published many papers in important journals and proceedings of influential conferences. He took part in commercial scientific projects related to data processing analysis and optimization. He teaches all subjects related to GPUs, including parallel data processing and computer graphics.

Pawel Boinski holds a PhD in Computing Science from Poznan University of Technology. In his research, he focuses on spatial data mining, especially discovery of co-locations and temporal patterns in spatial datasets. He participates in multiple projects related to data processing. Pawel authored papers that have been published in high-ranked journals, such as Information Sciences or Expert Systems with Applications. He primarily teaches subjects involving databases, data warehousing, internet applications, knowledge discovery and applications of computer science in e-society.