Geoinformatica (2016) 20:801–828
DOI 10.1007/s10707-016-0255-0

Mining spatiotemporal co-occurrence patterns in non-relational databases

Berkay Aydin[1] · **Vijay Akkineni**[1] · **Rafal Angryk**[1]

Received: 15 December 2014 / Revised: 29 November 2015 /
Accepted: 29 March 2016 / Published online: 13 April 2016

Abstract Spatiotemporal co-occurrence patterns (STCOPs) represent the subsets of feature types whose instances are frequently co-occurring both in space and time. Spatiotemporal co-occurrences reflect the spatiotemporal *overlap* relationships among two or more spatiotemporal instances both in spatial and temporal dimensions. STCOPs can be potentially used to predict and understand the generation and evolution of different types of interacting phenomena in various scientific fields such as astronomy, meteorology, biology, geosciences. Meaningful and statistically significant data analysis for these scientific fields requires processing sufficiently large datasets. Due to the computationally expensive nature of spatiotemporal operations required for mining spatiotemporal co-occurrences, it is increasingly difficult to identify spatiotemporal co-occurrences and discover STCOPs in centralized system settings. As a solution, we developed a cloud-based distributed mining system for discovering STCOPs. Our system uses Accumulo, a column-oriented non-relational database management system as its backbone. In order to efficiently mine the STCOPs, we propose three data models for managing trajectory-based spatiotemporal data in Accumulo. We introduce an in-memory join-index structure and a join algorithm for effectively performing spatiotemporal join operations on spatiotemporal trajectories in non-relational databases. Lastly, with the experiments with artificial and real life datasets, we evaluate the performance of the proposed models for STCOP mining.

Keywords Spatiotemporal Pattern mining · Non-relational databases

✉ Berkay Aydin
baydin2@cs.gsu.edu

Vijay Akkineni
vakkineni1@cs.gsu.edu

Rafal Angryk
rangryk@cs.gsu.edu

[1] Department of Computer Science, Georgia State University, 25 Park Place, Suite 700, Atlanta, GA 30303, USA

1 Introduction

Discovering interesting, but implicit spatiotemporal patterns from datasets is important for many scientific domains such as astronomy, ecology, meteorology, public health, agricultural sciences [15]. The ever-growing nature of data being generated and collected from various scientific sources makes the data-driven knowledge discovery process very challenging to the researchers in these fields. The spatiotemporal co-occurrence patterns (STCOPs) can be used for modeling various scientific phenomena (e.g., tornadoes, propagation of epidemics, clouds). The patterns can be utilized for performing large-scale verification of current knowledge, as well as the prediction of unknown spatiotemporal relationships among different types of feature (i.e. event) types (e.g., prediction the spread of epidemics [20], verification of hurricane landfall precipitation models [14], discovery of the patterns in wildlife migration [16], prediction of blastocyst formation [31]). One important application area for our applied research is solar physics. Spatiotemporal co-occurrences frequently transpire among solar events such as active regions and sunspots. Identifying STCOPs appearing on the surface of the Sun can help us better understand the relationships among solar event types and lead to better modeling and forecasting of crucial events such as solar flares and coronal mass ejections. These solar events can affect the radiation in space; they can impact the safety of space and air travel, and even damage power grids [21].

Spatiotemporal co-occurrence is a specific kind of generic movement patterns, which represents the relationships among the objects that frequently coincide both in space and time [4]. The mixed-drove spatiotemporal co-occurrence patterns represent spatiotemporal feature types whose (point-based) instances are located in spatial and temporal proximity [12]. Spatiotemporal co-occurrence patterns from evolving regions is interested in the discovery of the subsets of feature types whose (region-based) instances overlap in both space and time [25]. While various kinds of spatiotemporal co-occurrence patterns exist in the literature (See Section 3 for more information), we are specifically interested in the spatiotemporal co-occurrences of moving instances with continuously evolving polygon-based representations [25]. In the context of this paper, we will use spatiotemporal co-occurrence pattern (STCOP) for referring to the spatiotemporal co-occurrence patterns discovered from datasets with evolving region-based representations, unless otherwise stated.

1.1 Motivation

In recent years, we have introduced new algorithms and novel techniques for the discovery of spatiotemporal co-occurrence patterns. The spatiotemporal co-occurrence pattern mining is introduced in [25], and a brute-force Apriori-based solution is presented. In [24], the STCOP mining algorithm is improved using a filter-and-refine approach, where insignificant co-occurrences are efficiently eliminated using a filter that utilizes OMAX significance measure. In addition to that, two spatiotemporal trajectory indexing techniques are employed for faster data retrieval in the spatiotemporal co-occurrence pattern mining process to further improve the efficiency of the process [7, 8]. The recent research work on STCOP mining presents an efficient mining schemata for conventional single machine systems [7, 24].

Big spatiotemporal data poses an enormous set of challenges regarding analytics, data processing, capacity, and validation [2]. As pointed out in [30], with the rate at which spatiotemporal data is being generated, it is necessary to develop efficient distributed systems

and analytical infrastructure. The applicability of current spatiotemporal co-occurrence mining systems to many real life datasets is very limited, and can further be improved using distributed settings. There are two major challenges of the centralized solutions specifically related to STCOP mining: (1) massive data transfers are required when accessing or updating the data, and (2) expensive geometric calculations are necessary when performing topological spatiotemporal operations. To overcome these issues, we present a scalable solution for mining spatiotemporal co-occurrence patterns in a distributed environment. Accumulo,[1] a non-relational and distributed database management system, is used for storage and retrieval of spatiotemporal data. One particular challenge we have encountered is the spatiotemporal data modeling in non-relational databases. Key-value stores (primarily used in non-relational databases) provide potentially advantageous schema-free storage; however, they lack the functionality of object-relational database management systems.

1.2 Scope

In this work, we have developed a distributed spatiotemporal co-occurrence pattern mining system. Conceptually, the implemented STCOP mining algorithm is similar to the state-of-art STCOP-miner algorithm presented in [8, 24]. The system uses a non-relational distributed database system as its backend. To integrate the STCOP-miner algorithm into a non-relational database ecosystem, we have designed spatiotemporal data models to be used in column-oriented databases. We also introduce data access and update algorithms along with an in-memory join-index structure that can handle big spatiotemporal data. The spatiotemporal data models are designed to exploit the distributed data storage and retrieval mechanisms of non-relational databases. On the other hand, data access and update algorithms with the indexing structure provide necessary spatiotemporal querying functionality for mining STCOPs in non-relational database systems.

Readers should note that the scope of this paper is limited to discovering spatiotemporal co-occurrence patterns from instances with evolving polygon-based representations, and other co-occurrence pattern mining algorithms are not considered. The STCOP mining algorithm uses a spatiotemporal join operation for identifying the co-occurrences. The join operation is a part of the Apriori-based candidate generation procedure. The frequent pattern growth-based or join-less approaches are not considered in the scope of this work.

1.3 Outline

The rest of this paper is organized as follows. In Section 2, to familiarize the reader with the terminology, the basic concepts related to Accumulo database are demonstrated. Moreover, we present the system architecture and discuss the applicability of the distributed mining schema to spatiotemporal data mining. In Section 3, previous work on different types of spatiotemporal co-occurrence patterns are discussed. In Section 4, the preliminary concepts for spatiotemporal co-occurrence pattern mining are demonstrated, and the STCOP mining algorithm is presented. In Section 5, we describe our methodology for discovery of STCOPs in distributed settings and introduce the spatiotemporal data models for non-relational databases. Moreover, we provide the algorithmic details of our distributed system. We demonstrate our experimental results in Section 6. In Section 7, we present future work and conclude the paper.

[1] The Apache Accumulo - https://accumulo.apache.org/

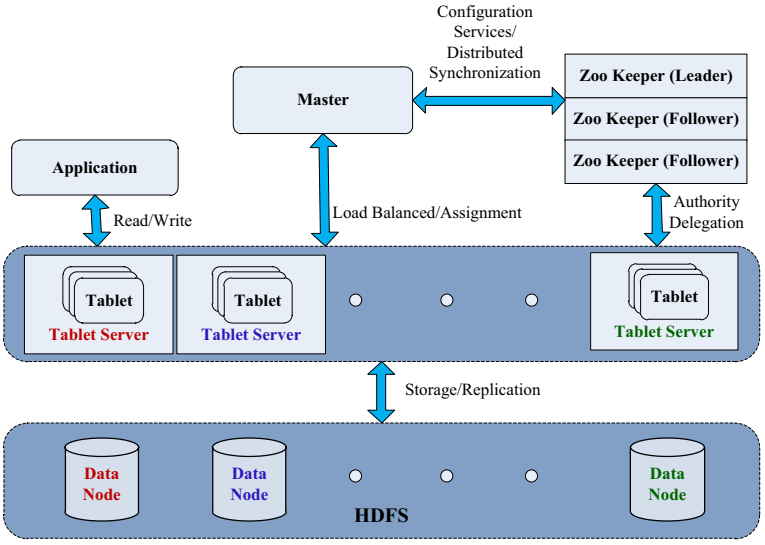

Fig. 1 Architectural overview of Accumulo

2 Non-relational databases for mining spatiotemporal data

Accumulo was inspired by Google's BigTable data storage system [13]. BigTable is a compressed, high performance and proprietary data storage system primarily built on *Google File System* (GFS) [17], *Chubby Lock Service* [9], and *Log-structured merge-tree* [23]. BigTable's architecture has been adopted by many popular non-relational databases such as HBase,[2] Cassandra,[3] and Accumulo. Accumulo provides high levels of consistency with scales to thousands of nodes and petabytes of data, and processes data in near real-time. We will briefly present the architecture behind our choice of Accumulo, and the suitability of its architecture for spatiotemporal data mining.

2.1 Architecture

Accumulo is a sparse, distributed, sorted and multi-dimensional key-value storage system that depends on Apache Hadoop Distributed File System (HDFS) for data storage and Apache Zookeeper for configuration [9, 27]. HDFS is designed to store very large files, with streaming data access running on a cluster of commodity hardware. It provides high throughput access to application data, and is pertinent for very large datasets. HDFS provides the clients with a single file system view to hide the underlying collection of data blocks spread across multiple data nodes.

The key components of the Accumulo architecture, shown in Fig. 1, are master, tablet servers, garbage collector, logger and monitor. The main function of *master* is to monitor the cluster for the status of tablet servers, assign tablets (partition of tables) to tablet servers, and

[2]HBase– http://hbase.apache.org/

[3]Cassandra– 'http://cassandra.apache.org/

perform load balancing. *Tablet server* component is responsible for handling all the reads and writes for the tables. In a typical deployment, one tablet server is colocated with one HDFS data node. Tablet server is registered with Accumulo's software by obtaining a lock from Zookeeper, and failures and recovery processes are handled by the master. Another task assigned to tablet servers is handling minor and major compactions. Minor compaction is the process of flushing the data stored in memory to sorted files stored on disk. Major compaction is merging these sorted files into a bigger file. Additional components (not shown in the Fig. 1) include: (1) *garbage collector* that deletes obsolete files from the file system, (2) *monitor* that is used for monitoring the key metrics of system resources used by Accumulo, (3) *logger* used for tracing the system events.

2.2 Applicability to spatiotemporal data mining

Some of the features of Accumulo, such as table partitioning, load balancing, and horizontal scalability, provide necessary tools for efficiently performing data mining tasks on spatiotemporal datasets. In this part of our discussion, we will briefly point to the important features of Accumulo to provide insight into the necessity of a distributed schema for spatiotemporal data mining from very large datasets.

The key strength of the data representations provided in Accumulo is the ability to store sparse multidimensional data. One practical feature of Accumulo database is *automatic table partitioning*, where tables are split after crossing a pre-configured row count threshold. Using this feature, tables can be stored across multiple tablet servers evenly, and the system can provide parallelized data access. *Load balancing*, provided by Accumulo, evenly spreads the workload of tablets to tablet servers, and ensures that tablet servers are not overloaded. Load balancing is important when implementing a distributed system for scalability purposes (e.g., avoiding hotspots in spatiotemporal data analysis by distributing the workload). Furthermore, *horizontal scalability* (also known as scaling out, i.e. adding more nodes to the system for increasing the workload capacity) can be achieved using Accumulo. In contrast to traditional object-relational databases used for spatiotemporal data in centralized settings, the scaling is cheaper in the means of economic cost [5]. Note that traditional databases should be scaled vertically (scaling up), meaning it is required to increase the system resources such as memory and CPU. Another noteworthy functionality of Accumulo database is the server side iterators. The main function of the iterators is concurrent traversal over the data with optional filtering or transformation. *Server side iterators* can offload some of the computations to the tablet servers, and lead to significant performance increases.

3 Related work on spatiotemporal co-occurrence patterns

Spatiotemporal co-occurrence pattern mining is conceptually similar to classical frequent pattern mining from transactional databases. However, the implicit spatial and temporal semantics (specifically spatial and temporal overlap) are required to be identified, and the identification of these relationships dramatically increase the complexity of the STCOP mining algorithms. In spatiotemporal frequent pattern mining, the underlying spatiotemporal semantic relationships are the main subjects of discovery. One of these relationships is the co-occurrence relationship, and it is originated from the significance of closeness in spatial and temporal domains, by asserting objects located in space and time proximity are more related than the others [28].

One pioneering advancement in spatial data mining is the discovery of spatial colocation patterns [29]. The spatial closeness of the objects is introduced as the *colocation* relationship. Given a set of boolean spatial features, spatial colocation mining aims to discover the subsets of features whose instances are frequently colocated together. As a matter of course, it is often very hard to observe point-based spatial objects sharing the same locations. Therefore, a neighborhood relationship (based on user specified thresholds) is used for defining the colocations. The colocation mining algorithm uses an Apriori-based approach [3], which requires a spatial join algorithm while generating and pruning the candidate patterns. Partial-join and join-less approach for mining colocations were presented in [32, 33].

While colocation refers to the purely spatial closeness of objects, the term *co-occurrence* is more frequently used for spatiotemporal closeness. Mixed-drove spatiotemporal co-occur-rence patterns (MDCOP) are introduced in [12]. MDCOPs represent the subsets of spatiotemporal feature types whose point-based instances are frequently occurring in spatial and temporal proximity. MDCOP-mining algorithms presented in [12] can be interpreted as a temporal extension of spatial colocation mining algorithms to spatiotemporal context. The proposed MDCOP-Miner algorithms follow a similar Apriori-based approach. Following mixed-drove spatiotemporal co-occurrence patterns, sustained emerging (SECOP) [12], partial (PACOP) [10], and periodical (PECOP) [11] spatiotemporal co-occurrence patterns are introduced. Fundamentally, emerging, partial, and periodical co-occurrence relationships are quite similar to MDCOPs. They include additional constraints, and require new interest measures tuned for these constraints. SECOPs represent the subsets of feature types whose instances are increasingly colocated in space and time. PACOPs are concerned with the discovery of spatiotemporal co-occurrences that are partially present in the database. PECOPs represent the subsets of feature types that are periodically co-occurring.

Spread patterns of spatiotemporal co-occurrences over zones (SPCOZ) are introduced in [26]. SPCOZs represent the subsets of feature types whose instances are spreading and co-occurring over particular zones. The main purpose of the mining SPCOZs is discovering spreading structures that co-occur together both in space and time (meaning correlations among the spreading structures are mined instead of trajectories). Another instance of spatiotemporal co-occurrence pattern mining is composite spatiotemporal co-occurrence (COSTCOP) [34]; where a new composite prevalence measure (using spatial and temporal dimensions together) is developed, and a pruning technique is developed for improving the performance of the mining algorithm.

Aforementioned spatiotemporal co-occurrence or colocation models were originally designed for instances with point-based geometric representations. As point-based instances exhibit nearly imperceptible spatial and temporal overlap relationships among each other, the spatial and temporal neighborhoods are to be defined for characterizing co-occur-rences or colocations. However, in spatiotemporal co-occurrence pattern mining from evolving region instances (defined over polygon data type), it is highly likely to observe spatial and temporal coincidences (namely spatiotemporal overlap relationships). Mining spatiotemporal co-occurrence patterns from datasets with evolving regions was introduced in [25]. The spatiotemporal instances, which are represented by polygons evolving over time, are treated as three-dimensional continuous objects. For deciding whether an overlap among these three-dimensional structures form a significant co-occurrence, a spatiotemporal version of Jaccard significance measure is used. Similar to the other co-occurrence

patterns, an Apriori-based algorithm (including a spatiotemporal join over spatial and temporal overlap predicates) is used. In [24], a novel filter-and-refine strategy for pruning the instances in the spatiotemporal join phase using OMAX measure is proposed. This algorithm is further improved in [8] by utilizing trajectory-based spatiotemporal indexing techniques.

One of the most remarkable observations about the past research work in co-occurrence pattern mining is considerably small datasets being used for testing the algorithms. Another observation we have made is the rapid increase in the runtime when using relatively larger datasets. Given these observations, one can apprehend that such data analyses tasks, when conducted on massive real life datasets, could greatly benefit from adapting an environment containing distributed storage and computational resources. In our work, we have used a cloud-based distributed database setting for storing and querying data. We have used a slightly modified version of STCOP mining algorithm [8] in non-relational database settings. The details of the algorithms will be presented in Section 4 and Section 5.

4 Preliminary concepts on spatiotemporal co-occurrence pattern mining

As mentioned earlier, we utilize the spatiotemporal co-occurrence pattern mining algorithm introduced in [8, 24], which mines the evolving region data established upon an Apriori-based algorithm that effectively prunes the candidates using a filter-and-refine strategy. Given a set of spatiotemporal feature types and spatiotemporal instances associated with these feature types, a spatiotemporal co-occurrence pattern is a subset of all features, whose instances frequently overlap both in temporal and spatial dimensions. Following parts of this section include the basic definitions, significance measures and the general algorithm for STCOP mining algorithm.

4.1 Definitions

Definition 1 A spatiotemporal **instance**, denoted as $Inst$, is a spatiotemporal object represented as a two-dimensional region continuously evolving over time. Each instance is identified with a unique identifier, denoted as *InstanceId*, and has a feature type associated with it. Instances have start and end times corresponding to birth and death times of the objects. For each valid timestamp, instances can have exactly one polygon-based geometric representation. The set of all instances is denoted by $\mathbb{I}$, and the set of all instances of a particular feature type (f_i) is denoted as $\mathbb{I}_{f_i}$.

Definition 2 A **feature type** (or event type) is a non-spatiotemporal attribute of an instance that signifies the sort (or class) of the instance. A feature type is denoted as f_i, and the set of all features is denoted by $\mathbb{F} = \{f_1, f_2, \ldots, f_m\}$.

Definition 3 A **spatiotemporal co-occurrence pattern** (referred to as **pattern** in this paper) is a subset of all feature types, whose instances frequently co-occur in both space and time. A pattern is denoted as P, where $P = \{f_{i_1}, \ldots, f_{i_k}\}$ and $P \subset \mathbb{F}$. The number of feature types in a pattern will be referred as the *cardinality* of the pattern. A k-cardinality pattern refers to a k-subset of all feature types, $\mathbb{F}$. The minimum cardinality of the pattern is 2 ($k \geq 2$).

Definition 4 Given a k-cardinality pattern $P = \{f_{i_1}, \ldots, f_{i_k}\}$, a **pattern instance** (denoted by $PInst$) of P is a unique incident of a spatiotemporal co-occurrence (namely, spatial and temporal overlap) among the instances of the *all* feature types in P. Similar to the instances, pattern instances have start and end times. The start time of a pattern instance corresponds to the minimum start time of the participating (co-occurring) instances. The end time of a pattern instance corresponds to the maximum end time of the participating instances. For each timestamp between the start and end time (lifetime), a pattern instance has two kinds of spatiotemporal representation. The first representation is designated for spatiotemporal *intersection* showing the overlapping regions over time for the participating instances while the second one is for spatiotemporal *union* demonstrating the union of all regions over the lifetime of the pattern instance. In Fig. 2, we illustrate two overlapping spatiotemporal instances ($Inst_i$ and $Inst_j$) and the pattern instance ($PInst_k$) generated from these instances. The start time of $PInst_k$ is t_1, and the end time is t_{10}. The intersection geometries of the $PInst_k$ are valid between the interval (t_3, t_4) while the union geometries of $PInst_k$ are valid between the interval (t_1, t_{10}).

4.2 Measures

We utilize two types of interestingness measures in our algorithm. The first one is the co-occurrence coefficient (cce), which is used for assessing the strength of the spatiotemporal overlap in a pattern instance. The second one is the prevalence measure (p), and it is used for evaluating the prevalence of a pattern.

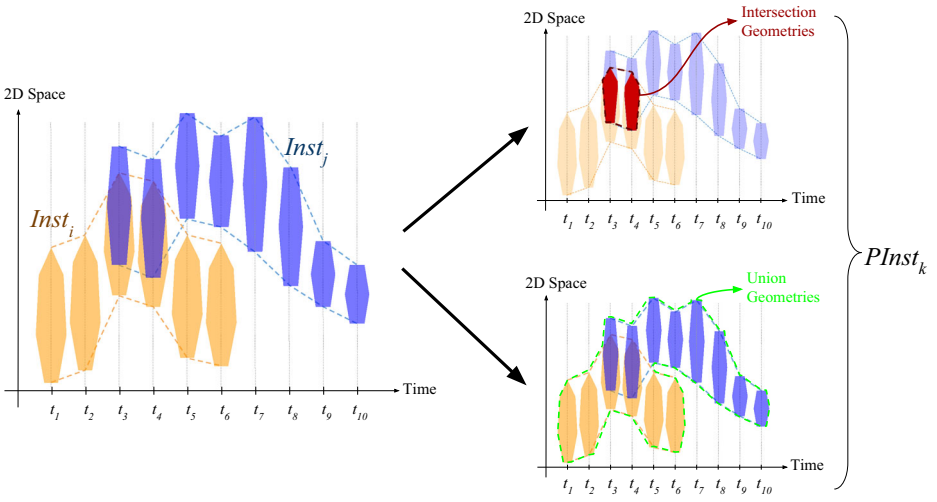

Fig. 2 The creation of a pattern instance ($PInst_k$) from two spatiotemporal instances ($Inst_i$ and $Inst_j$). On the *left*, the spatiotemporal instances are illustrated. On the *right*, the intersection (*top*) and union (*bottom*) geometries of the pattern instance are demonstrated

4.2.1 Significance measures

The spatiotemporal co-occurrence coefficient is used for determining the strength of an overlap (both in space and time) relationship. To assess the strength of spatiotemporal overlap, we utilize the commonly used *Jaccard (J)* measure as co-occurrence coefficient. Additionally, the computationally cheaper $OMAX$ measure is utilized for filtering.

The J measure for a pattern instance is calculated as the ratio of spatiotemporal intersection volume to the spatiotemporal union volume. Given a k-cardinality pattern P, and the pattern instance, $PInst$, and let $\{Inst_1, \ldots, Inst_k\}$ be the participating instances of $PInst$. Then, the J measure is calculated as follows:

$$J = \frac{V(Inst_1 \cap \ldots \cap Inst_k)}{V(Inst_1 \cup \ldots \cup Inst_k)} \tag{1}$$

V is the spatiotemporal volume function. The spatiotemporal instances and pattern instances are three-dimensional objects in two spatial and one temporal dimensions. The spatiotemporal intersection operation is represented with $\cap$, and the spatiotemporal union operation is represented with $\cup$.

On the other hand, the $OMAX$ measure for a pattern instance is calculated as the ratio of spatiotemporal intersection volume to the maximum volume of all the participating instances. Following the notation used for the J measure (in Eq. 1), $OMAX$ measure is calculated as,

$$OMAX = \frac{V(Inst_1 \cap \ldots \cap Inst_k)}{Max(V(Inst_1), \ldots, V(Inst_k))} \tag{2}$$

where Max function returns the largest participating instance volume. The $OMAX$ measure contains J measure. In other words, maximum volume of participating instances is less than or equal to the union volume of all participating instances ($Max(V(Inst_1), \ldots, V(Inst_k) \leq V(Inst_1 \cup \ldots \cup Inst_k)$). Therefore, for any pattern instance, the $OMAX$ value is always greater than or equal to the J value. Due to the containment property, given a specific *cce* threshold, for each pattern instance (*PInst*) passing Jaccard (J) filter, it is guaranteed that it also passes $OMAX$ filter ($OMAX(PInst) \geq J(PInst)$). Proof of this property can be found in [24].

4.2.2 Prevalance measure

In STCOP mining, for assessing the interestingness of a pattern, the participation index is used. The participation index signifies the prevalence of a pattern [18]. For a k-cardinality spatiotemporal co-occurrence pattern, the participation index (denoted as $pi(P)$) is defined as follows:

$$pi(P) = Min_i^k \, pr(P, f_i) \tag{3}$$

where $P = \{f_{i_1}, \ldots, f_{i_k}\}$ and $f_i \in \mathbb{F}$. Let $|P_{f_i}|$ denote the number of unique instances of feature type f_i participating in the pattern instances of P, and $|\mathbb{I}_{f_i}|$ is the total number of instances of feature type f_i; then, the participation ratio of a feature type f_i in the pattern P is,

$$pr(P, f_i) = \frac{|P_{f_i}|}{|\mathbb{I}_{f_i}|} \tag{4}$$

Prevalent spatiotemporal co-occurrence patterns are characterized by a co-occurrence coefficient threshold (cce_{th}) for determining the significant pattern instances of the pattern, and prevalence measure threshold (p_{th}) for assessing the interestingness of the pattern. For a prevalent pattern, the co-occurrence coefficient for all of its pattern instances must be greater than or equal to cce_{th} and the participation index for the pattern must be greater than or equal to p_{th}.

Algorithm 1 STCOP-Miner Algorithm

Input: A spatiotemporal dataset with instances of different feature types, co-occurrence coefficient (cce_{th}) and prevalance threshold (p_{th})
Output: The set of prevalent STCOPs denoted as FP

```
1: function STCOP-MINER(cce_th, p_th)
2:     ⟨𝕀, 𝔽⟩ ← READDATASET()
3:     k ← 1    \\ k represents cardinality
4:     FP[k] ← 𝔽
5:     while FP[k] ≠ ∅ do
6:         CP ← GENERATECANDIDATEPATTERNS(FP[k])
7:         CP ← GENERATEPATTERNINSTANCES(CP, cce_th, p_th)
8:         k ← k + 1
9:         FP[k] ← CP
10:    end while
11:    return FP − FP[1]
12: end function
```

4.3 STCOP-miner algorithm

The pseudocode of the STCOP mining algorithm can be seen in Algorithm 1. The algorithm follows greedy Apriori iterations to discover the patterns (See Lines 5 to 10 in Algorithm 1). The input of the algorithm is a dataset containing instances of different feature types. The spatiotemporal instances have evolving polygon-based representations. The result returned by the algorithm is a list of prevalent spatiotemporal co-occurrence patterns.

The initialization step of the algorithm creates the database, and reads the dataset from input files (See Line 1 in Algorithm 1). The variable k, which is the index representing the size of the patterns is set to 1 (See Line 2 in Algorithm 1). Two-dimensional list of patterns, FP, represents the prevalent patterns. The index of the list represents the cardinality. In other words, $FP[k]$ points to a list of k-cardinality patterns. For the conciseness and correctness of the algorithm, we store feature types in the first index of FP (See Line 4 in Algorithm 1). While the set of feature types may be a necessary output for some application areas, our actual prevalent STCOP set is the difference $FP - FP[1]$.

After the initialization steps, the algorithm follows the Apriori-based steps, where firstly candidate patterns are generated, and later pruned. The candidate patterns are generated using GENERATECANDIDATEPATTERNS procedure, and stored in CP, which is a list of patterns (See Line 6 of Algorithm 1). The prevalent k-cardinality patterns found in the previous iteration ($FP[k]$) is self-joined ($FP[k] \times FP[k]$), and $(k+1)$-cardinality candidates which have infrequent subpatterns are removed from CP. Using the candidate patterns (CP), the candidate pattern instances are generated in the procedure GENERATEPATTERNINSTANCES. (See Line 7 of 1). GENERATEPATTERNINSTANCES procedure initially identifies the significant pattern instances, and eliminates the infrequent candidate patterns.

Algorithm 2 Candidate Pattern Instance Generation with Filter-and-Refine Procedure

Input: The list of k-cardinality candidate patterns (CP), co-occurrence coefficient (cce_{th}) and prevalance threshold (p_{th})
Output: The list of k-cardinality prevalent STCOPs, whose significant pattern instances are identified and written back to database

```
1:  procedure GeneratePatternInstances(CP, cce_th, p_th)
2:      for all pattern P ∈ CP do
3:          ⟨L1, L2⟩ ← GetSubPatterns(P)    \\ L1, L2 ∈ P are two minimal k-cardinality patterns
4:          CPInst ← ST_Join(L1, L2, Overlap)    \\ CPInst is found by joining L1 and L2
                                                 \\ based on spatiotemporal overlap predicate.
5:          for all pattern instance PInst ∈ CPInst do
6:              patOMAX ← calculateOMAX(PInst)
7:              if patOMAX < cce then
8:                  Remove PInst ffrom CPInst
9:              end if
10:         end for
11:         piOMAX ← calculatePI(CPInst)
12:         if piOMAX < p_th then
13:             Remove P from CP
14:             continue
15:         end if
16:         for all pattern instance PInst ∈ CPInst do
17:             patJ ← calculateJaccard(PInst)
18:             if patJ < cce_th then
19:                 Remove PInst
20:             end if
21:         end for
22:         piJ ← calculatePI(CPInst)
23:         if piJ < p_th then
24:             Remove P from CP
25:             continue
26:         else
27:             WriteBack(CPInst)    \\ write the significant pattern instances back to database
28:         end if
29:     end for
30:     return CP
31: end procedure
```

In Algorithm 2, the pseudocode of candidate pattern instance generation procedure (GeneratePatternInstances) is demonstrated. The pattern instances are generated and pruned for each candidate pattern (See Lines 2 to 21 in Algorithm 2). For each candidate pattern (P, let the cardinality of P be k), firstly, two $(k-1)$-cardinality subpatterns are determined, and the pattern instances of those two subpatterns are joined with a spatiotemporal overlap predicate and candidate patterns instances are generated and stored in $CPInst$ list. (See Lines 3 and 4 in Algorithm 2). It is important to note that the subpatterns of any generated candidate pattern (P) must be prevalent; therefore, their pattern instances are stored in the database. The spatiotemporal join operation is handled in ST_Join procedure. For each candidate pattern in $CPInst$, initially, the $OMAX$ value is computed, then if it cannot pass the co-occurrence coefficient threshold (cce_{th}), the pattern instance is removed from $CPInst$ (See Lines 5 to 10 in Algorithm 2). After filtering the pattern instances with the $OMAX$ measure, the participation index is calculated for the candidate pattern (See Line 11 in Algorithm 2). Note that some of the candidate pattern instances may already be removed. Then, if the candidate pattern cannot pass the given participation index threshold

(pi_{th}), the candidate pattern is removed from the candidate pattern list (CP) (See Lines 12 to 15 in Algorithm 2). If it can pass, the J value is calculated for filtered pattern instances. Similar to the $OMAX$ filtering, the J values are calculated for remaining candidate pattern instances and the insignificant ones are removed from the list. When calculating the J values, we only calculate union volumes, as intersection volumes are already calculated in $OMAX$ filtering steps. Using the remaining candidate pattern instances, the participation index is calculated (See Lines 16 to 21 in Algo. 2), and if the candidate cannot pass the participation index threshold, it is removed from the list (See Lines 23 to 25 in Alg. 2); else, the significant candidate pattern instances are written back to the database (See Line 26 in Alg. 2).

It is important to note that the most vital part of this procedure is the spatiotemporal join operation based on the spatiotemporal overlap predicate (See Line 4 of Algorithm 2 – ST_JOIN($L_1, L_2, Overlap$)). We will explain different implementations of the join procedure in Section 5, and for the integrity of our research, we use a generic nested loop based join algorithm for every data model. See Algorithm 3 for a detailed view. The spatiotemporal search procedure (SEARCH-ST-OVERLAP) in the generic join algorithm differs for each data model. These search algorithms can be found in Section 5.

Algorithm 3 Generic Spatiotemporal Join Algorithm

Input: Two tables L_1 and L_2 which are containing instances
Output: The result of spatiotemporal join, *Results*

```
1: function SPATIOTEMPORALJOIN(L1, L2)
2:     for all Inst ∈ L1 do
3:         OverlappingInstances ← SEARCH-ST-OVERLAP(Inst, L2)
4:         for all OInst ∈ OverlappingInstances do
5:             Add ⟨Inst, OInst⟩ to Results
6:         end for
7:     end for
8:     return Results
9: end function
```

5 Modeling spatiotemporal co-occurrences in non-relational databases

The data in Accumulo is represented as simple key-value pairs. However, Accumulo provides moderately richer data modeling opportunities compared to simple key-value stores [1]. The *key* field in Accumulo is comprised of a row identifier, column identifier, and timestamp. Column identifier has three values: column family, column qualifier and column visibility (See Fig. 3). In our data models, we only used row identifier, column family and column qualifier for identifying the values. Column visibility and timestamp values are not used.

Key					**Value**
Row Id	*Column*			*Timestamp*	
	Family	Qualifier	Visibility		

Fig. 3 The elements of a key-value pair in Accumulo

In the following parts of this section, the data models for storing spatiotemporal instances are presented. All the data models use the generic join algorithm presented in Algorithm 3. The spatiotemporal search procedure is different for each data model. The search algorithms of the data models are presented in their respective sections.

5.1 Classical data model

The first data model we have designed is the classical data model (CDM) where each row stores one spatiotemporal instance or pattern instance. This data model mimics the models used in relational database settings. In Fig. 4, we demonstrate the hierarchical decomposition of a row (which stores an instance (a) or a pattern instance (b)) in CDM. The row identifiers (Row Id) are the instance identifiers (or pattern instance identifiers). Column family points to feature types and column qualifiers show timestamps. Value field represents the polygon of an instance in each timestamp. In addition to those, for more efficient spatial and temporal filtering when accessing the instances, metadata fields are generated. Metadata fields store temporal and spatial boundaries of instances and pattern instances. Temporal boundaries are the start and end time of instances or pattern instances. Spatial boundaries are the minimum bounding rectangle of the union of instance geometries or the minimum bounding rectangle of the union of pattern instance's intersection geometries. In CDM, one instance is stored in a row comprised of multiple columns. Each column in the row is uniquely identified by the timestamp value. The columns point to a particular polygon representing the spatial extension of the instance at a valid timestamp. For pattern instances, the timestamps (in column qualifiers) are divided into two groups: intersection and union geometries. Intersection timestamps point to the intersection geometries of the pattern instances while union timestamps point to the union geometries.

The algorithm of spatiotemporal overlap search procedure for an instance, which is stored using the classical data model, is presented in Algorithm 4. The input parameters for the procedure are the query instance object ($QInst$), and the table name, L, where the search (or scan) will be performed. In the initialization part, the minimum bounding rectangle (MBR), start time, and end time of query instance are fetched from the database. Also, a server-side scan iterator, which is used for scanning the entire table (from 0 to ∞ for row identifiers) is also initialized. (See Line 5 in Algorithm 4 - GETITERATOR.) Then, for each instance ($Inst$), which is returned by the iterator ($Iter_L$), temporal filter and spatial filter are applied using the metadata fields. If the lifetimes (start and end times) of $QInst$ and $Inst$ overlap, and their MBRs intersect, then the actual polygon representations of those two instances are checked. If the actual polygons also overlap (both spatially and temporally),

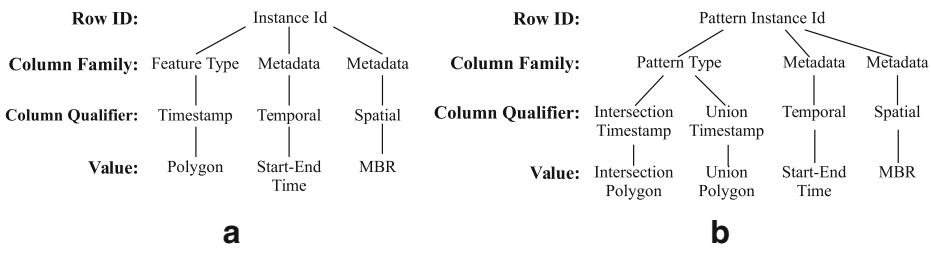

Fig. 4 The hierarchical decomposition of key-value pairs as an instance **(a)** and a pattern instance **(b)** in classical data model (CDM)

then $Inst$ is added to the result set ($SResults$). See Lines 7 to 17 in Algorithm 4. After the entire table, L, is scanned, the procedure returns the result set, $SResults$.

Algorithm 4 Spatiotemporal Search Algorithm for CDM

Input: The query instance *QInst* and table *L* containing instances
Output: The result of spatiotemporal search based on spatial and temporal overlap predicates, *SResults*

```
 1: function SEARCH-ST-OVERLAP(QInst, L)
 2:     Mbr_QInst ← GETMBR(QInst)
 3:     STime_QInst ← GETSTARTTIME(QInst)
 4:     ETime_QInst ← GETENDTIME(QInst)
 5:     Iter_L ← GETITERATOR(L, 0, ∞)    \\ returns an iterator scanning entire table
 6:     Inst ← Iter_L.NEXT()
 7:     while Inst ≠ NULL do
 8:         if TEMPORALLYINTERSECTS(Inst, STime_QInst, ETime_QInst) then    \\ temporal filter
 9:             if MBRINTERSECTS(Inst, Mbr_QInst ) then    \\ spatial MBR filter
10:                 if ISOVERLAPPING(QInst, Inst) then
11:                 \\ ISOVERLAPPING() returns true if there exists an overlap on actual polygons
12:                     Add Inst to SResults
13:                 end if
14:             end if
15:         end if
16:         Inst ← Iter_L.NEXT(null)
17:     end while
18:     return SResults
19: end function
```

5.2 Spatiotemporal data model

As mentioned earlier, the data in Accumulo is sorted based upon the row identifiers; however, the column identifiers do not carry such a property. With spatiotemporal data model, we intend to make use of the ordered nature of row identifiers. In spatiotemporal data model (STDM), we followed a three-dimensional space-driven partitioning strategy. The partitioning space consists of two spatial dimensions (denoted by x and y) and one temporal dimension (denoted by t), and it is divided into non-overlapping three-dimensional cells. The instances are considered as three-dimensional trajectories spanning through these cells. To divide the partitioning space into cells, three user-defined step-size parameters are used. Each parameter indicates the step size of their respective dimensions, and denoted by Δx, Δy, and Δt. Our space partitioning algorithm can be seen in Algorithm 5. In a nutshell, the partitioning algorithm iteratively determines the time partition (See Line 4 of Algorithm 5) and space partitions (See Lines 5 to 17 of Algorithm 5) of each polygon at a valid timestamp.

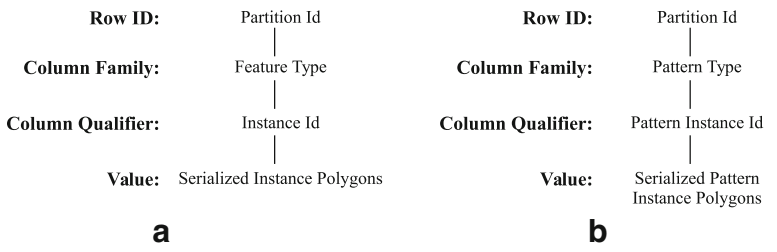

Fig. 5 The hierarchical decomposition of key-value pairs as an instance **(a)** and a pattern instance **(b)** in spatiotemporal data model (STDM)

Algorithm 5 Space Partitioning Algorithm

Input: The input instance, *Inst*, and the step sizes, Δx, Δy, and Δt
Output: The set of 3D cells *Inst* spans - CellSet

```
1: function PARTITION(Inst, Δx, Δy, Δt)
2:     CellSet ← {}
3:     for all ⟨Timestamp_i, Polygon_i⟩ pairs ∈ Inst do
4:         t_p ← ⌊Timestamp_i/Δt⌋
5:         ⟨x_min, y_min, x_max, y_max⟩ ← GETMBRCOORDINATES(Polygon_i) \\ returns MBR's minimum and
6:                                                                       \\ maximum x and y coordinates
7:         for i = ⌊x_min/Δx⌋ to ⌊x_max/Δx⌋ do
8:             cellX_min ← i.Δx and cellX_max ← (i+1).Δx
9:             for j = ⌊y_min/Δy⌋ to ⌊y_max/Δy⌋ do
10:                cellY_min ← j.Δy and cellY_max ← (j+1).Δy
11:                CellRectangle ← CREATERECTANGLE((cellX_min, cellY_min), cellX_max, cellY_max))
12:                if INTERSECTS(CellRectangle, Polygon_i) then
13:                    Cell ← [t_p, i, j]
14:                    Add Cell to CellSet
15:                end if
16:            end for
17:        end for
18:    end for
19:    return CellSet
20: end function
```

In STDM, the row identifier is the spatiotemporal partition cell identifier, while the column qualifier shows the instance (or pattern instance) identifier. The hierarchical decomposition of instances and pattern instances in STDM is shown in Fig. 5. For pattern instances, the partition cells are calculated for only intersection geometries. The entire list of timestamp–polygon pairs are serialized and stored in the *value* field. By setting row identifier as the spatiotemporal partition cell, we aim to retrieve possibly co-occurring instances more efficiently by exploiting the order enforced by Accumulo. For the instances (or pattern instances) which span through more than one partition cells, a duplication strategy is used. Namely, the instances may be inserted to database more than once. The rows can store more than one instance (or pattern instance) in STDM. Each column (identified by the instance or pattern instance identifier) of a row points to a different instance or pattern instance.

Algorithm 6 Spatiotemporal Search Algorithm for STDM

Input: The query instance *QInst* and table *L* containing instances
Output: The result of spatiotemporal search based on spatial and temporal overlap predicate *SResults*

```
1: Initializaton: Join algorithm performs a brute force search for QInst in the database
2: function SEARCH-ST-OVERLAP(QInst, L)
3:     PartitionCells ← PARTITION(QInst, Δx, Δy, Δt)
4:     for all cellId ∈ PartitionCells do
5:         Iter_L ← GETITERATOR(L, cellId, cellId)   \\ returns an iterator scanning only one
6:                                                   \\ row specified by cellId from table L
7:         Inst ← Iter_L.NEXT()
8:         while (Inst ≠ NULL) do
9:             if ISOVERLAPPING(QInst, Inst) then
10:                Add Inst to SResults
11:            end if
12:            Inst ← Iter_L.NEXT()
13:        end while
14:    end for
15:    return SResults
16: end function
```

The algorithm of spatiotemporal overlap search procedure for STDM can be seen in Algorithm 6. The algorithm takes a query instance (denoted as $QInst$) and a table name as its parameter. The initialization step of the search algorithm (See Line 1 in Algorithm 6) plays an important role when evaluating the runtime efficiency of this procedure. As all the instances are written to the database, an initial querying of the database is unavoidable when performing this particular search algorithm. After the initialization step, the procedure primarily determines the spatiotemporal partition cells (denoted as $PartitionCells$) of query instance ($QInst$) (See Line 3 in Algorithm 6). The server-side scan iterators for searching the database are prepared using the cell identifiers (denoted as $cellId$) in $PartitionCells$ set (See Line 5 in Algorithm 6). Note that these iterators search for only one row, which is specified by their partition identifier; namely $cellId$. For each instance returned by the iterators, the spatiotemporal overlap predicate is checked, and if they overlap, the instance is added to the result set ($SResults$). See Lines 7 to 14 in Algorithm 6. After iterating over all partition cells, the procedure returns the result set, $SResults$.

5.3 Indexed spatiotemporal data model

Indexed spatiotemporal data model (ISTDM) is an extension to the spatiotemporal data model. ISTDM employs an in-memory join-index structure, which stores the partition cells of each instance (or pattern instance). For mapping the partition cells to instances, an inverted index structure [22] is used. Traditionally, the inverted index is used for text retrieval where each index entry (a word) points to the documents where the queried word occurs. The index entries for our case are the locations (identified by partition cell identifiers) of instances or pattern instances. For each feature type (or pattern), an inverted index is created, while their instances are getting written to the database. An example scenario is demonstrated in Fig. 6. For a particular feature type (f_i), the instances are $\mathbb{I}_{f_i} = \{Inst_1, Inst_2, Inst_3, Inst_4, Inst_5\}$. Spatiotemporal partition cells are denoted by $STPartition_i$. Instances can be part of multiple partition cells. For example, $Inst_4$ is only in $STPartition_1$, while $Inst_3$ spans through $STPartition_1$, $STPartition_5$, and $STPartition_6$. The row identifiers used for storing instances in the database are the partition cell identifiers. On the other hand, in the inverted index, the row key is the instance identifier, and each instance identifier is mapped to the partition cell identifiers.

ISTDM and STDM use the same hierarchical decomposition of key-value pairs for instances and pattern instances. However, the spatiotemporal overlap search procedure is slightly modified for faster access. Algorithm 7 shows the search procedure for the indexed spatiotemporal data model. The main difference is in the initialization section. The STDM

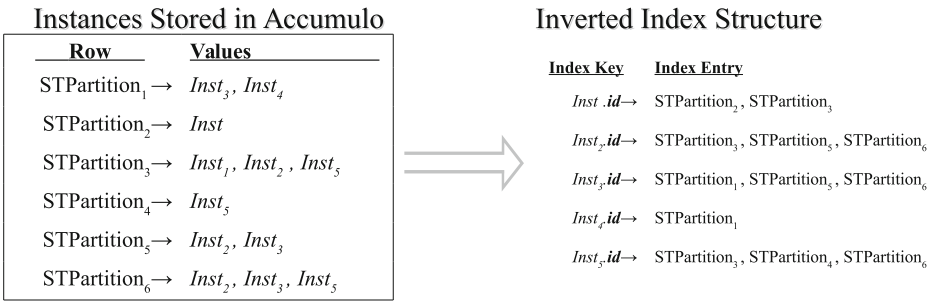

Fig. 6 Example: The creation of inverted index structure for a particular feature type

search algorithm (Algorithm 6) performs a brute-force search to locate the query instance in the database in the initialization step. However, IFSTDM search algorithm uses our inverted index structure to determine the partition cells, and use it to fetch $QInst$ and the possibly co-occurring instances in table L. Also, $PartitionCells$ set is not calculated, as we fetch this information from the index. Therefore, ISTDM eliminates initial brute-force search and the determination of partition cells for the query instance.

Algorithm 7 Spatiotemporal Search Algorithm for ISTDM

Input: The partition cells, which query instance *QInst* span through, *PartitionCells*, and table *L* containing instances

Output: The result of spatiotemporal search based on spatial and temporal overlap predicate *SResults*

```
1: Initializaton: Join algorithm searches QInst.id in inverted index and returns PartitionCells
                  Using a cellId ∈ PartitionCells, we find and retrieve QInst from database
2: function SEARCH-ST-OVERLAP(QInst, PartitionCells, L)
3:     for all cellId ∈ PartitionCells do
4:         Iter_L ← GETITERATOR(L, cellId, cellId)
5:         Inst ← Iter_L.NEXT()
6:         while (Inst ≠ NULL) do
7:             if ISOVERLAPPING(QInst, Inst) then
8:                 Add Inst to SResults
9:             end if
10:            Inst ← Iter_L.NEXT()
11:        end while
12:    end for
13:    return SResults
14: end function
```

6 Experimental evaluation

For analyzing our data models and the search algorithms designed for these models, we developed a distributed cloud-based STCOP mining system and experimented with it using different datasets and database settings. Our primary intent in these experiments is to observe the effect of data models and using a distributed database setting to the runtime performance of our STCOP mining system. In the following parts of this section, the experimental settings with the datasets will be described, the implementation details of our system will be demonstrated, and the results and analysis of our experiments will be given.

6.1 Experimental settings

We used six artificial and five real life datasets in our experiments. The artificial datasets are created using spatiotemporal dataset generator, ERMO-DG [6]. Two of the real life datasets are the solar event datasets, and the remaining three are the basketball datasets. All of our datasets are publicly available in our website.[4]

The six artificial datasets are named as *A, B, C, D, E,* and *F*. The average lifetime of the instances is 12.5 (minimum = 10, maximum = 15), and it is kept the same for all datasets for the ease of analysis. The noise ratio used in all datasets are 4.0. The smallest artificial dataset (dataset *A*) has 16,799 polygons (containing 113,720 point references in total), while

[4] http://grid.cs.gsu.edu/~baydin2/proj/nonrelstcop.html

Table 1 Datasets used in our experiments and their basic properties

Domain of dataset	Dataset name	Number of feature types	Total number of polygons	Total number of vertices in polygons
Artificial	A	9	16,799	113,720
Artificial	B	9	33,538	214,106
Artificial	C	9	70,131	468,124
Artificial	D	9	140,238	963,575
Artificial	E	9	280,949	1,883,763
Artificial	F	9	562,439	3,391,507
Solar Event	3Mo	6	480,136	21,711,503
Solar Event	6Mo	6	922,323	47,445,474
Basketball	ATL1	7	440,527	19,889,980
Basketball	ATL2	7	486,086	21,939,703
Basketball	ATL3	7	505,001	22,798,527

the largest one (dataset *F*) has more than 562,439 polygons (containing 3,391,559 point references in total). All artificial datasets have nine artificially created feature types.

For solar event datasets, the geometric representations of solar event instances are downloaded using the Web API of Heliophysics Event Knowledgebase,[5] and tracked and interpolated using the algorithm described in [19]. The solar datasets contain the instances of six different solar event types that are Active Regions, Coronal Holes, Emerging Flux, Filaments, Sigmoids, and Sunspots. The three-month solar event dataset (denoted as *3Mo*) contains solar event instances between '01/07/2013' to '30/09/2013'. The six-month solar event dataset (denoted as *6Mo*) contains solar event instances between '01/01/2013' to '30/06/2013'.

The basketball datasets are obtained from the NBA's official statistics page.[6] We scraped the data for three games of Atlanta Hawks in 2014-2015 season, which are following: (1) 'Atlanta Hawks – Toronto Raptors' in '29/10/2014' (denoted as *ATL1*), (2) 'Atlanta Hawks – Los Angeles Clippers' in '05/01/2015' (denoted as *ATL2*), (3) 'Washington Wizards – Atlanta Hawks' in '04/02/2015' (denoted as *ATL3*). The raw data from NBA contains the point locations of the basketball and ten players on the court for a particular period (a specific play) of a game. The specified point locations of the players are buffered to create polygons. We categorized the players based on their teams (Atlanta Hawks (Atl) or the opponent (Opp)), and the positions of players (i.e., center (C), forward (F), guard (G)). Therefore, including the basketball, we have seven feature types that are Ball, Atl-C, Atl-F, Atl-G, Opp-C, Opp-F, and Opp-G.

The properties of artificial and real life datasets can be seen in Table 1. The total number of points (vertices) in the polygons can be seen in the last column.

One goal of our experiments is to observe the effect of horizontal scaling (scaling out) in our system. In order to see the effects of scaling out, we conducted our experiments using one, three and six tablet servers hosting the database. As data nodes are colocated with tablet servers, the term tablet server is used instead of data nodes. The experiments using one tablet

[5]Heliophysics Event Registry - https://www.lmsal.com/hek/api.html

[6]NBA.com/Stats - http://stats.nba.com/

Table 2 The system settings for experiments using three (3TS) and six (6TS) tablet servers

3TS-nodes	Assigned roles	6TS-nodes	Assigned roles
Node 1	NameNode	Node 1	NameNode
Node 2	ZooKeeper Leader	Node 2	ZooKeeper Leader
Node 3	Secondary NameNode	Node 3	Secondary NameNode
Node 4	Accumulo Master	Node 4	Accumulo Master
Node 5,6,7	Tablet Server and Data Node	Node 5,6,7,8,9,10,11	Tablet Server and Data Node

server is included for exhibiting an experimental environment showing very similar characteristics to traditional database settings. We ran the three and six tablet server experiments on Amazon Web Services cloud computing platform.[7] Because of the economic constraints, we ran the one tablet server experiments on a local virtual machine. The system settings when using three and six tablet servers can be seen in Table 2. Nodes that are assigned to the role *NameNode* controls the distributed file system (HDFS). *Zookeeper leader* coordinates the ZooKeeper *followers* (that are Secondary NameNode and Accumulo master) for input and output operations. *Secondary NameNode* role is an auditing system for Hadoop, performing periodical checkpoints. *Accumulo master* is responsible for load balancing, as well as error detection, in tablet servers. The *tablet server* and *data node* roles are colocated for decreasing the network latency. Tablet servers are responsible for managing (i.e., reads and writes) a subset of all tables. Data nodes simply store data in HDFS. In one tablet server settings, all the above-mentioned roles are performed on the same machine.

In one tablet server experiments, a virtual machine in a personal computer (with 2Ghz Intel Core i7 CPU and 4GB memory and 64 GB SSD storage) is used for conducting the experiments. In three and six tablet server experiments, the nodes used in AWS, are medium size computing instances (officially listed as *m3.large*). These instances include 10-core 2.5 GHz Intel Xeon E5-2670 CPUs, 7.5 GB memory, and 64 GB SSD storage.

6.2 Implementation details

The STCOP mining system is implemented as a Java client connecting to the Accumulo database. JTS Topology Suite[8] library is employed for performing geometric operations. The geometries are stored using the well-known text (WKT) format. In the experiments with artificial datasets, co-occurrence coefficient threshold (cce_{th}) is set to 0.01 and prevalence measure threshold (p_{th}) is set to 0.01. For solar event and basketball datasets, cce_{th} is set to 0.001 and p_{th} is set to 0.05. We increased the p_{th} for the experiments with real life datasets to keep the counts of generated patterns and pattern instances comparable among the datasets.

In Section 2, we have explained the automatic table splitting feature provided in Accumulo. Using automatic table splitting, a table can be split and distributed into multiple partitions when the number of rows in the table passes a certain threshold (which is provided by the administrator of the database). Another technique used for partitioning the data is *pre-splitting*. When pre-splitting is employed, a number of splitting points (let n be the number of splitting points) are provided to database upon creation, and tables are split and

[7] Amazon Web Services - Cloud Computing Services – http://aws.amazon.com/

[8] JTS Topology Suite – http://www.vividsolutions.com/jts/JTSHome.htm

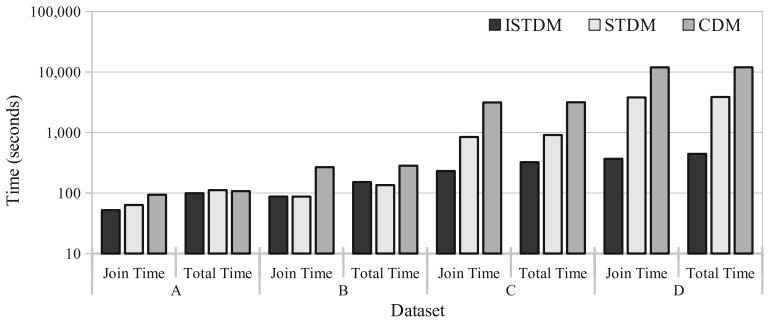

Fig. 7 The total runtime and time spent on joins for one tablet server experiments

distributed into $(n + 1)$ partitions based on these splitting points. Balanced partitioning of data is important as the number of active search iterators can be adequately increased when tables are split across the different data nodes. Pre-splitting is more advantageous when compared to automatic table partitioning, as pre-splitting can decrease the data transfer times when splitting the tables, and efficiently utilize parallel server-side search iterators.

6.3 One tablet server experiments

In one tablet server experiments, we analyzed the runtime performance of STCOP mining algorithm using classical (CDM), spatiotemporal (STDM), and indexed spatiotemporal (ISTDM) data models. For these experiments, we employed relatively smaller artificial datasets: *A*, *B*,*C*, and *D*. We run the experiments using one tablet server and one scanner (search iterator). In Fig. 7 the total runtime and time spent on spatiotemporal join operations are shown in logarithmic scale, and in Fig. 8 the size of the database is shown. The total runtime (shown as *Total Time* in our charts) is the total time spent for running the STCOP-Miner algorithm shown in Algorithm 1. The time spent on spatiotemporal join operations (shown as *Join Time* in our charts) is calculated as the sum of the total time spent on locating the query instances and performing spatiotemporal overlap search. Note that the spatiotemporal joins are known to be the performance bottleneck of the STCOP miner algorithm [7].

From Fig. 8, it can be observed that CDM has less storage requirements. As STDM and ISTDM use the same storage model, the size of the database is the same for STDM

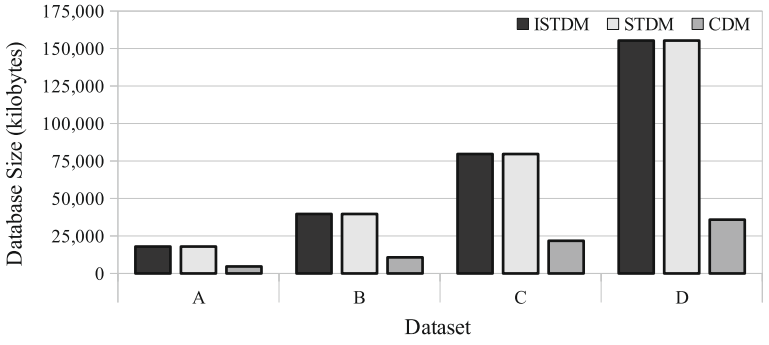

Fig. 8 Total database sizes of datasets for one tablet server experiments

and ISTDM. CDM uses a string representation (WKT) of polygon objects, and timestamps are encoded in the column qualifier field of keys. On the other hand, STDM and ISTDM perform serialization on each spatiotemporal instance and store them as byte arrays. As we use a simplistic Java-based serialization model without any compression, the increase in the storage requirements for STDM and ISTDM is understandable. Another factor contributing to this increase is the duplication strategy used (when instance trajectories span across more than one partition cells, the instance is stored in multiple partition cells). We also observed that from 11 % to 37 % of instances are duplicated at least once when running one tablet server experiments.

While CDM provides compact data storage, spatiotemporal data models (STDM and ISTDM) provide better runtime performance. For smaller datasets, the difference in the runtime is not visible. For the dataset *A*, the total running time of CDM is even less than STDM, while the IFSTDM achieves only 1.08x speedup. However, the difference in join times is apparent when we inspect the experimental results from larger datasets (i.e., *B*, *C*, and *D*). By using STDM (when compared to CDM), we can see from 2.09x speedup for dataset *B* to 3.51x speedup for dataset *C* in one tablet server settings. Furthermore, using indexing on top of STDM (in ISTDM) provides better total runtime. ISTDM achieves up to 8.7x speedup when compared to STDM, and 27x speedup when compared to CDM.

6.4 Multiple tablet server experiments

In multiple tablet server experiments, we employ the Amazon Web Services cloud computing platform with three and six tablet servers as shown in Table 2. The STCOP mining system is tested with larger size artificial, solar event, and basketball datasets. In multiple tablet server experiments, our focus is to demonstrate the horizontal scalability of the system and our data models with different dataset characteristics.

6.4.1 Experiments with artificial datasets

For multiple tablet server experiments, relatively larger artificial datasets *C*, *D*, *E*, and *F* were tested. Based on our findings from one tablet server experiments with smaller artificial datasets, we did not conduct the experiments with CDM due to the poor scalability of CDM as shown in Fig. 7. We test the capabilities of CDM in the experiments with real life datasets.

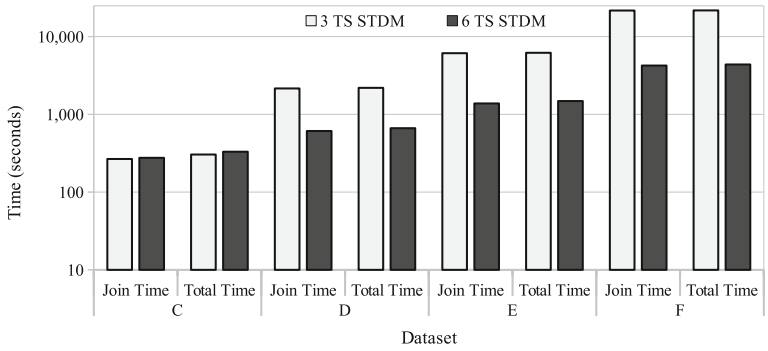

Fig. 9 The total runtime and time spent on joins for the artificial datasets *C, D, E*, and *F* in three and six tablet server settings

Figures 9 and 10 show the total runtime of STCOP-Miner algorithm and the time spent on spatiotemporal joins for STDM and ISTDM in three (shown as *3 TS*) and six (*6 TS*) tablet server settings. Using multiple tablet servers and data nodes, we aimed to show the effects of horizontal scalability on database side for our STCOP mining system.

Firstly, from Fig. 9, we can see the differences of using more data nodes in STDM. Only for the relatively smaller dataset *C*, the three tablet server setting performs better than six tablet server setting for STDM. We observe 3.3 to 5.0x speedup when we increase the number of tablets from three to six. On the other hand, for ISTDM (shown in Fig. 10), we do not see the effect of increasing the number of tablet servers. However, this is expected as our inverted index provides direct access to key values of instances or pattern instances. Both query instance and the instances in search results can be accessed by a constant number of lookups. Therefore, we are not able to see particularly significant speedups when using ISTDM. For STDM parallel scan iterators significantly change the time spent on searching the instances. Additionally, ISTDM performs better than STDM for almost all datasets.

6.4.2 Experiments with solar datasets

The results of the multiple tablet server experiments with the solar event datasets (*3Mo* and *6Mo*), are shown in Fig. 11. Three tablet server experiment results are shown with plain bars, and six tablet server experiment results are shown with striped bars. It is important to mention that the instances in artificial datasets have shorter lifespans and simpler geometries while the instances in solar datasets have unbalanced data characteristics and significantly more complex geometries. Another important difference between the artificial and solar datasets is the number of generated candidate pattern instances. Solar datasets generate less candidate pattern instances; therefore, the join and total runtime for solar datasets are shorter than artificial datasets.

We can observe that STDM and ISTDM perform significantly better than the CDM for both datasets, and the difference between the total runtimes of data models is caused mainly by the difference in the join times. We also notice the performance increase when six tablet servers are used. The performance of the joins in CDM has increased when using six tablet servers (in *3Mo* by 9 % and in *6Mo* by 21 %). Similarly, for ISTDM, the join performance has increased for both datasets, as well as the total runtime. For STDM in *6Mo* dataset, we recognize an 11 % performance drop. This can be explained by the overhead

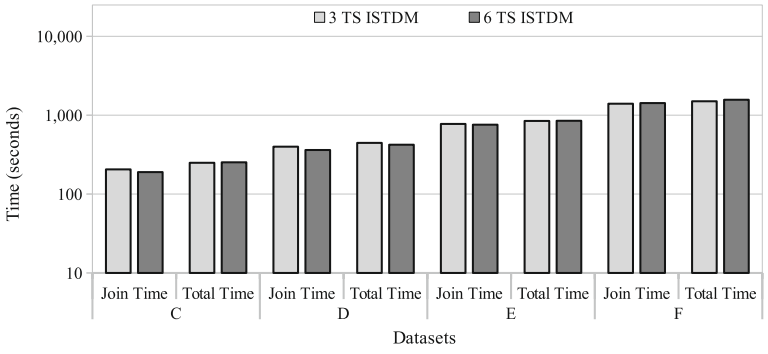

Fig. 10 The total runtime and time spent on joins for the artificial datasets *C, D, E*, and *F* in three and six tablet server settings

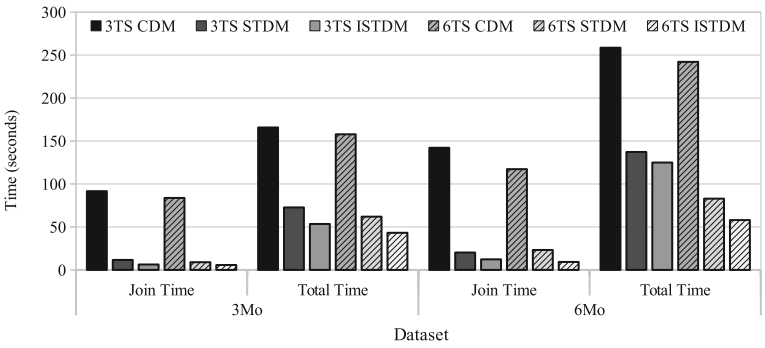

Fig. 11 The time spent on spatiotemporal joins (Join Time) and the total runtime of STCOP-Miner algorithm (Total time) for the solar event datasets *3Mo* and *6Mo* in three (3TS) and six (6TS) tablet server settings using classical (CDM), spatiotemporal (STDM), and indexed spatiotemporal (ISTDM) data models

of search iterator creation. Nevertheless, the total runtime performance for STDM (in *6Mo*) has increased because of the parallelized writers.

6.4.3 Experiments with basketball datasets

The results of the multiple tablet server experiments with the basketball datasets (*ATL1*, *ATL2* and *ATL3*), are shown in Fig. 12. Similar to the experiments with the solar event datasets, three tablet server experiment results are shown with plain bars, and six tablet server experiment results are shown with striped bars. Instances in basketball datasets have significantly longer lifespans when compared to artificial datasets. Additionally, the geometries of the instances in the basketball datasets are simpler than the ones in solar event datasets. Similar to the solar event datasets, the number of generated candidate pattern instances for the basketball datasets is smaller for the artificial datasets. However, the lifespans of the pattern instances are longer.

The experimental results from the basketball datasets are similar to the results from the solar event datasets. Both STDM and ISTDM perform better than CDM. Using six tablet servers leads to a performance increase in join times, as well as the total runtimes for all the

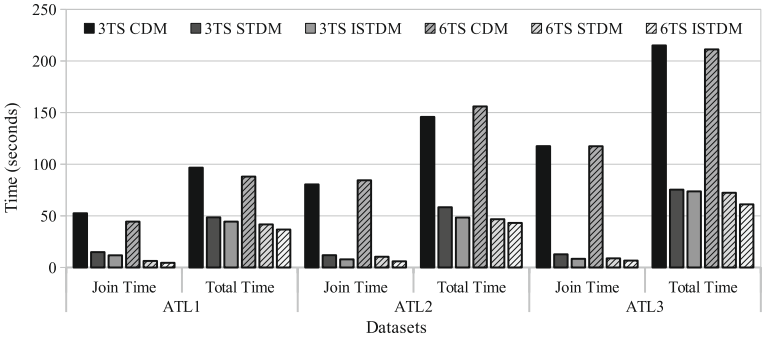

Fig. 12 The time spent on spatiotemporal joins (Join Time) and the total runtime of STCOP-Miner algorithm (Total time) for the basketball datasets *ATL1*, *ATL2* and *ATL3* in three (3TS) and six (6TS) tablet server settings using classical (CDM), spatiotemporal (STDM), and indexed spatiotemporal (ISTDM) data models

settings. In addition to that, indexing (used in ISTDM) accelerates the join operations for all the datasets.

6.5 Remarks

In a nutshell, we can affirm that ISTDM provides better runtime performance than STDM and CDM for large spatiotemporal datasets, and the total runtime performance of STCOP mining system is heavily affected by the spatiotemporal join performance.

The artificial, solar event and basketball datasets have significantly different characteristics, and our experiments with all of them provide better insight into the behavior of the mining system under different scenarios. Firstly, we have observed the highest speedup from CDM to ISTDM in the artificial datasets (up to 27x). This can be explained with the higher number of generated candidate pattern instances from the artificial datasets.

Another goal of our experiments was to inspect the scalability of our proposed data models. In the STCOP mining system, when multiple tablet servers are used, data access and update operations are performed in parallel. The distribution of data access and update workload increases the performance of the system for all the data models. The spatiotemporal join performance of STDM and ISTDM is not significantly improved when more tablet servers are used. However, the total runtime performance of these models is better when more tablet servers are used, as we write the pattern instances back to the database in parallelized fashion. Especially, in the solar event dataset experiments, we observe the performance increase caused by the parallelized write operations with six tablet servers for STDM. This is clearly visible from *6Mo* dataset STDM results in Fig. 11, where the join time with six tablet servers is more than the join time with three tablet servers; however, the total runtime for six tablet server is shorter, as the parallelized writes significantly increase the runtime performance.

7 Conclusion and future work

One integral part of knowledge discovery is the efficient retrieval of data. For relational databases, data access methods are well-defined and efficient retrieval techniques are facilitated by the database vendors. However, in the context of spatiotemporal databases, efficient and effective access methods are not available because of the diversity of spatiotemporal data (historical, predicted, moving objects, etc.), complex representations (points, line strings, polygons, geometry collections, etc.) and rich semantics (temporal sequences, spatial colocations, spatiotemporal co-occurrences etc.). With the increasing volumes of spatiotemporal data, it becomes necessary to employ distributed databases when mining big spatiotemporal data. In this work, we designed a distributed STCOP mining system which employs a distributed non-relational database, Accumulo, for storing spatiotemporal instances. We have introduced data models to store spatiotemporal instances in column-oriented non-relational databases. These models are classical (CDM), spatiotemporal (STDM) and indexed spatiotemporal (ISTDM) models. Classical data model simulates the object-relational database modeling. Spatiotemporal data model follows a space-driven partitioning strategy to capture the implicit spatial and temporal information, and exploits the sorted nature of keys in Accumulo for efficient data retrieval. Indexed spatiotemporal data model uses the same data modeling on the database as STDM, and utilize an inverted index structure for improving the spatiotemporal search and join performance of STCOP mining.

For testing the performance of the distributed STCOP mining system, we conducted experiments with three proposed models and eleven datasets under different distribution settings. In our experiments, we consistently see that ISTDM performs the best in the means of runtime performance. On the other hand, CDM does not perform well mainly because it does not capture and take advantange of the implicit spatiotemporal information, which leads to higher spatiotemporal join times. However, it is worth mentioning that CDM has less storage requirement than STDM and ISTDM. We also observe that using more computing nodes increases the performance of the system for all data models.

In the future, one potential problem to investigate is the join-less mining schemas for discovering STCOPs in distributed database environments. Another interesting problem is the frequent pattern growth-based approaches for the task of STCOP mining.

Acknowledgments This work was supported in part by two NASA Grant Awards (No. NNX11AM13A, and No. NNX15AF39G), and one NSF Grant Award (No. AC1443061). The NSF Grant Award has been supported by funding from the Division of Advanced Cyberinfrastructure within the Directorate for Computer and Information Science and Engineering, the Division of Astronomical Sciences within the Directorate for Mathematical and Physical Sciences, and the Division of Atmospheric and Geospace Sciences within the Directorate for Geosciences.

References

1. Apache Accumulo user manual version 1.6., https://accumulo.apache.org/1.6/accumulo_user_manual.html (2014). Accessed: December 1, 2014
2. Agouris P, Aref W, Goodchild MF, Barbra S, Jensen J, Knoblock CA, Langley R, Mikhail E, Shekhar S, Wolfson O, Yuan M (2012) From GPS and virtual globes to spatial computing-2020. Tech. rep., Computing Community Consortium
3. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules in large databases. In: VLDB'94, Proceedings of 20th international conference on very large data bases, Santiago de Chile, pp 487–499
4. Andrienko NV, Andrienko GL (2007) Designing visual analytics methods for massive collections of movement data. Cartographica 42(2):117–138
5. Armbrust M, Fox A, Griffith R, Joseph AD, Katz RH, Konwinski A, Lee G, Patterson DA, Rabkin A, Stoica I, Zaharia M (2010) A view of cloud computing. Commun ACM 53(4):50–58
6. Aydin B, Angryk RA, Pillai KG (2014) ERMO-DG: Evolving region moving object dataset generator. In: Proceedings of the twenty-seventh international florida artificial intelligence research society conference, FLAIRS 2014, Pensacola Beach
7. Aydin B, Kempton D, Akkineni V, Angryk R, Pillai KG (2015) Mining spatiotemporal co-occurrence patterns in solar datasets. Astronomy and Computing. doi:10.1016/j.ascom.2015.10.003. In Press
8. Aydin B, Kempton D, Akkineni V, Govaparam S, Pillai KG, Angryk R (2014) Spatiotemporal indexing techniques for efficiently mining spatiotemporal co-occurrence patterns. In: Workshop on solar astronomy big data, 2014 IEEE International Conference on Big Data. IEEE, pp 1–10
9. Burrows M (2006) The Chubby lock service for loosely-coupled distributed systems. In: Proceedings of the 7th symposium on operating systems design and implementation 2006, OSDI '06. USENIX Association, Seattle, pp 335–350
10. Celik M. (2011) Discovering partial spatio-temporal co-occurrence patterns, Fuzhou, pp 116–120
11. Celik M., Azginoglu N., Terzi R. (2012) Mining periodic spatio-temporal co-occurrence patterns: a summary of results. In: 2012 international symposium on innovations in intelligent systems and applications (INISTA), pp 1–5
12. Celik M, Shekhar S, Rogers JP, Shine JA (2008) Mixed-drove spatiotemporal co-occurrence pattern mining. IEEE Trans Knowl Data Eng 20(10):1322–1335
13. Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE (2008) Bigtable: a distributed storage system for structured data. ACM Trans Comput Syst 26(2)
14. Elsberry RL (2002) Predicting hurricane landfall precipitation: optimistic and pessimistic views from the symposium on precipitation extremes. Bull Am Meteorol Soc 83(9):1333–1339

15. Erwig M (2004) Toward spatio-temporal patterns. In: de Caluwe R, de Tr G, Bordogna G (eds) Spatio-temporal databases. Springer, Berlin, pp 29–53
16. Gauthreaux SA, Belser CG (2003) Bird movements on Doppler weather surveillance radar. Birding 35(6):616–628
17. Ghemawat S, Gobioff H, Leung S (2003) The google file system, Bolton Landing, pp 29–43
18. Huang Y, Shekhar S, Xiong H (2004) Discovering colocation patterns from spatial data sets: a general approach. IEEE Trans Knowl Data Eng 16(12):1472–1485
19. Kempton D, Pillai KG, Angryk RA (2014) Iterative refinement of multiple targets tracking of solar events. In: 2014 IEEE international conference on big data, big data 2014, Washington, pp 36–44. doi:10.1109/BigData.2014.7004402, (to appear in print)
20. Kuhn K, Campbell-Lendrum D, Haines A, Cox J (2005) Using climate to predict infectious disease epidemics. World Health Organ, Geneva
21. Langhoff SR, Straume T (2012) Highlights of the space weather risks and society? workshop. Space Weather 10(6)
22. Manning CD, Raghavan P, Schütze H (2008) Introduction to information retrieval. Cambridge University Press
23. O'Neil PE, Cheng E, Gawlick D, O'Neil EJ (1996) The log-structured merge-tree (lsm-tree). Acta Inf 33(4):351–385
24. Pillai KG, Angryk RA, Aydin B (2013) A filter-and-refine approach to mine spatiotemporal co-occurrences. In: 21st SIGSPATIAL international conference on advances in geographic information systems. SIGSPATIAL, Orlando, pp 104–113
25. Pillai KG, Angryk RA, Banda JM, Schuh MA, Wylie T (2012) Spatio-temporal co-occurrence pattern mining in data sets with evolving regions. In: 12th IEEE international conference on data mining workshops, ICDM Workshops, Brussels, pp 805–812
26. Qian F, He Q, He J (2009) Mining spread patterns of spatio-temporal co-occurrences over zones. In: Computational science and its applications - ICCSA 2009, international conference. Proceedings, Part II, Seoul, pp 677–692
27. Sen R, Farris A, Guerra P (2013) Benchmarking apache accumulo bigdata distributed table store using its continuous test suite. In: IEEE international congress on big data. BigData Congress, pp 334–341
28. Shekhar S, Chawla S (2003) Spatial databases - a tour. Prentice Hall
29. Shekhar S, Huang Y (2001) Discovering spatial co-location patterns: A summary of results. In: Proceedings advances in spatial and temporal databases, 7th international symposium, SSTD 2001, Redondo Beach, pp 236–256
30. Vatsavai RR, Ganguly A, Chandola V, Stefanidis A, Klasky S, Shekhar S (2012) Spatiotemporal data mining in the era of big spatial data: Algorithms and applications. In: Proceedings of the 1st ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data, BigSpatial '12. ACM, New York, pp 1–10. doi:10.1145/2447481.2447482, (to appear in print)
31. Wong CC, Loewke KE, Bossert NL, Behr B, De Jonge CJ, Baer TM, Pera RAR (2010) Non-invasive imaging of human embryos before embryonic genome activation predicts development to the blastocyst stage. Nat Biotechnol 28(10):1115–1121
32. Yoo JS, Shekhar S (2004) A partial join approach for mining co-location patterns. In: Proceedings 12th ACM international workshop on geographic information systems, ACM-GIS 2004, Washington, pp 241–249
33. Yoo JS, Shekhar S (2006) A joinless approach for mining spatial colocation patterns. IEEE Trans Knowl Data Eng 18(10):1323–1337
34. Zhang Z, Wu W (2008) Composite spatio-temporal co-occurrence pattern mining. In: Proceedings of Wireless algorithms, systems, and applications, third international conference, WASA 2008, Dallas, pp 454–465

Berkay Aydin is currently a Ph.D. candidate at Department of Computer Science at Georgia State University, and a member of Data Mining Lab. He received his B.Sc. degree from Department of Computer Engineering at I.D. Bilkent University in 2012. His main areas of research interests are spatiotemporal pattern mining, data modeling, and information retrieval.

Vijay Akkineni is currently a doctorate student at Department of Computer Science at Georgia State University, and a member of Data Mining Lab. He received his M.Sc. degree from Department of Computer Science at Texas Tech University in 2007, and his B.Tech. degree from National Institute of Technology Warangal in 2003. His research interests are big data, data mining, and parallel computing.

Rafal Angryk Dr. Angryk is an associate professor of computer science, an affiliate professor of astronomy, and the director of Data Mining Lab at Georgia State University. He received M.S. and Ph.D. degrees in Computer Science in 2004 from Tulane University. His main research interests lie in the data mining area, and specifically in the challenge of new knowledge acquisition from real-life, massive databases. Dr. Angryk has published over 100 journal articles, book chapters and peer-reviewed conference papers in these areas. His research has been sponsored by the federal agencies: NASA, NGA, NSF and industry: Intergraph Corporation, RightNow Technologies (currently Oracle), with a successful grant history exceeding $10 M.