Journal of Intelligent Information Systems manuscript No.
(will be inserted by the editor)

Generalized Discovery of Tight Space-Time Sequences

Antonio Castro · Heraldo Borges · Riccardo Campisano · Fabio Porto · Reza Akbarinia · Florent Masseglia · Esther Pacitti · Rafaelli Coutinho · Eduardo Ogasawara

Received: DD/MM/YYYY / Accepted: DD Month YEAR

Abstract Spatiotemporal patterns bring knowledge about sequences of events, place and time in which they occur. Finding such patterns is a complex task and of great value for different domains. However, not all patterns are frequent across an entire dataset, being often constrained in space and time. This paper proposes the G-STSM algorithm to discover frequent sequences constrained in space and time, without using prior restrictions. This allows different sequence sizes, time ranges and space to be found, being the first to allow the use of space in three dimensions. G-STSM was tested using two real-world spatiotemporal datasets from health and seismic domain, showing its quality, performance and generality.

Keywords Pattern Mining · Spatio-temporal

1 Introduction

The popularization of digital devices with sensors and GPS contributes to the emergence of extensive databases with relevant sequences at a given time and space. Thus, space-time sequence mining has become increasingly important in

Antonio Castro, Heraldo Borges, Ricardo Campisano
CEFET/RJ, Federal Center for Technological Education of Rio de Janeiro,
E-mail: {antonio.castro, hborges, rcampisano}@eic.cefet-rj.br

Fabio Porto
LNCC, National Laboratory of Scientific Computing,
E-mail: fporto@lncc.br

Reza Akbarinia, Florent Masseglia, Esther Pacitti
INRIA, University of Montpellier,
E-mail: {reza.akbarinia,florent.masseglia,esther.pacitti}@inria.fr

Rafaelli Coutinho, Eduardo Ogasawara
CEFET/RJ, Federal Center for Technological Education of Rio de Janeiro,
E-mail: rafaelli.coutinho@cefet-rj.br, eogasawara@ieee.org

several domains (Alatrista-Salas et al., 2016; Li and Fu, 2014). The ability to analyze this data opens opportunities to extract interesting patterns (Koseoglu et al., 2020). However, the frequency of occurrence of some patterns is not always high throughout an entire dataset. Then comes the idea of extracting events that are frequent across a window of time and space.

Consider, for example, that downtown and main streets in Rio de Janeiro city on regular workdays present traffic jams before and after working hours. It is not different for the streets around the Maracanã stadium, where a large flow of vehicles also causes traffic jams. Due to its high support, a sequential pattern that could easily be found is composed of jams before and after working hours. They occur on the streets towards downtown before working hours and on the streets from downtown afterward.

However, when football matches take place at the Maracanã stadium on Wednesday nights. At the end of the match, around 10 pm, passenger transport cars begin to serve the fans who are leaving the stadium. It results in new traffic jams on the streets around the stadium (in all directions, from and towards the city center) propagating to Rodoviária do Rio bus terminal. They occur not only due to many cars but also because they move slowly, waiting for customers. This kind of pattern would hardly be found using traditional methods because, from a daily perspective, it has low support. The purpose of this work is precisely to find these sequences of events (proximity to the end of the soccer match generates traffic jam), the set of positions (streets close to the Maracanã stadium), and the time range (end of the soccer match) where this pattern is frequent. Knowing the existence of this pattern is of great importance for city planning and management.

An intuitive solution to find them is to search for frequent sequences of events with low support. After finding their occurrences, group them into clusters of sequences restricted in space and time. A possible implementation of this solution is to combine well-known algorithms for sequence mining and clustering. We might use, for example, classical SPADE (Zaki, 2001) and DBSCAN (Ester et al., 1996) together. This give us the implementation named SPADE + DBSCAN (Alatrista-Salas et al., 2015). However, it can generate a high number of frequent sequences due to the low support used, impacting the performance of the approach.

In this sense, different methods have been proposed for spatiotemporal data mining. Some use only data mining to search for frequent patterns, taking into account only the time (Li and Fu, 2014). Others combine techniques looking for frequent patterns in time and later group them in the space (Flamand et al., 2014). Furthermore, there is a diversity in the way restrictions are handled. Some use global support, a support value that is valid for the entire dataset (Alatrista-Salas et al., 2016). Others consider local support, making use of predefined time and space windows (Koseoglu et al., 2020).

This article explores the discovery of frequent sequences constrained in space and time, without using prior restrictions. The next paragraphs introduces the problem definition, the proposed approach for its solution and the main contributions of this work.

Problem definition: Considering a Spatial Time-stamped Sequence (STS) D, the problem addressed in this work is to find sequences in D that are frequent in constrained space and time. The goal is to discover frequent sequences, the time range, and the set of positions in which these sequences are frequent.

Approach and contributions: The present work presents a novel approach to tackle the problem. Data mining is performed by iteratively looking for frequent sequences in a time range and checks if they occur close in space, *i.e.*, in a group. For that, the user establishes three thresholds. First, a minimum frequency to be reached in a time range. Second, a maximum distance for each position in a group should satisfy its closest position inside it. Third, a lower limit of different positions inside the group. Thus, the formalization presented in this work can find different sizes of sequences, time ranges, and regions of space where a sequence is frequent.

As far as the investigations carried out has reached, the only work with a similar approach found in the literature is the one proposed by Campisano et al. (2018). Such work searches for frequent sequences but considers the space dimension linearly. The present work generalizes it presenting a formalization that considers space in its three-dimensional form. In contrast, the search for sequences is done in integrated time and space. However, it is done in a different order from that presented by Campisano et al. (2018). Two experiments were conducted using real-world spatio-temporal datasets to show the quality, performance, and generality of our algorithm. The experimental evaluation was done in different domains: seismic and health.

In addition to this introduction, this article is organized into five more sections. Section 2 discusses related work. Section 3 provides the Generalized Spatial-Time Sequence Miner (G-STSM) algorithm for the generalized discovery of tight space-time sequences. Section 4 presents the experimental evaluation. Finally, Section 5 concludes this article.

2 Related Works

A systematic map study was conducted on August 9^{th} of 2021, using the search string ("sequence mining" OR "sequential pattern") AND ("space-time" OR "spatiotemporal") in Scopus. The search considered the title, abstract, and keyword and was limited to articles in the English language. A total of 86 documents were retrieved. Some are unrelated to the topic discussed here. Two documents were a survey on pattern mining (Sukanya and Ranjit Jeba Thangaiah, 2019; Sunitha and Rama Mohan Reddy, 2014), and the others can be classified according to the dataset that they use: 41 trajectory analyses or 23 fixed position analyses.

Trajectory analysis describes datasets as a collection of events of the same object, or natural phenomena, moving in different time and space. Mobility patterns to find out best places to deploy Roadside Units to maintain vehicular ad hoc networks is an example (Ben Chaabene et al., 2021). It can be a human activity-travel (Xu and Kwan, 2020), moving polygon-based regions (Aydin and Angryk, 2016; Aydin et al., 2020), or even a taxi ride (Ibrahim and Shafiq, 2019; Yang and Gidófalvi, 2018; Cheng et al., 2020). Chen et al. (2014) and Lee et al. (2016) try to generate mobility human profiles, Huang et al. (2016) mine frequent trajectory patterns from posts in a social network, He et al. (2020) mine trajectory on complex geographic phenomena, and Feuerhake and Sester (2013) try to recognize group movement patterns. Although these works are relevant, they seek patterns different from those explored in this work. Thus, instead of trajectory patterns, we intend to discover patterns of sequences of events, where all events of the same sequence occur in the fixed position.

Fixed position analysis includes datasets that use sensors. They commonly acquire data from the surrounding environment without moving (Batu et al., 2017). These can be customer behavior (Chen et al., 2020; Koseoglu et al., 2020), hydrological data (Alatrista-Salas et al., 2015, 2016), satellite images (Julea et al., 2008, 2011) or even crime patterns (Chen et al., 2015, 2017).

Although this work is part of the fixed position analysis, each work deals with time and space constraints differently. Some use global support, *i.e.*, a support value that is used for the entire dataset (Tsoukatos and Gunopulos, 2001; Leong and Chan, 2012; Xue et al., 2016; Yusof and Zurita-Milla, 2017; Batu et al., 2017; Zhang et al., 2018). Some have local support, predefined restrictions on time, space, or both. They commonly establish a fixed grid size and search for sequences that are frequent inside that grid. Julea et al. (2011) limit space in number of pixels. Flamand et al. (2014) limit space in geographic territories and time in intervals. Gurram and Rama Mohan Reddy (2014) map events in the dataset from grid cells according to time and space sizes. Alatrista-Salas et al. (2015) pre-process data building homogeneous zones of spatial objects. Chen et al. (2015) limit space in radius and time in intervals. Sunitha and Rama Mohan Reddy (2016) limit space in sub-regions and time into intervals. Alatrista-Salas et al. (2016) limit space in zones. Chen et al. (2017) predefine a time range and a radius for space. Chen et al. (2020) consider time within a week to avoid large gaps between events. Koseoglu et al. (2020) enable the user limit both time and space. Our work deals with it differently. In our work, we do not start with time or space constraints. We find the frequent sequences, the region in space, and the range of time where they are frequent. It can find different time intervals and regions of space for each frequent sequence.

When mining spatio-temporal datasets, each work uses different methods. Some use methods that search for frequent patterns taking into account only time (Ben Chaabene et al., 2021; Li and Fu, 2014; Lee et al., 2016; He et al., 2020; Tsoukatos and Gunopulos, 2001; Julea et al., 2008; Leong and Chan, 2012; Xue et al., 2016; Batu et al., 2017; Zhang et al., 2018). Other introduce some prior restriction to also take space into account (Aydin and Angryk, 2016; Aydin et al., 2020; Julea et al., 2011; Flamand et al., 2014; Gurram and Rama Mohan Reddy, 2014; Chen et al., 2015; Sunitha and Rama Mohan Reddy, 2016; Chen et al., 2017; Koseoglu et al., 2020). Others use methods to search for frequent patterns over time and also clustering spatial positions (Feuerhake and Sester, 2013; Chen et al., 2014; Huang et al., 2016; Yusof and Zurita-Milla, 2017; Ibrahim and Shafiq, 2019; Alatrista-Salas et al., 2015, 2016; Chen et al., 2020). Our approach, again, takes a different approach. Our method searches for frequent sequences in time, but such frequent sequences must occur in spatial groups. Such groups have a certain minimum size (defined by the user). Each position in a group must be close to, at least, another position (below a maximum distance threshold) inside it.

Table 1 shows the classification of these related works according to the analysis, method, and constraints. The analysis includes division between trajectory or fixed position. The used method can be sequence mining or clustering with sequence mining. Finally, constraints are related to adopting global support or local support with fixed space or fixed time.

Table 1 Classification of related works relative to method used and constraints

Class	**Work**	**Method**		**Constraints**		
		Seq.	Clust.+ Seq.	Glob. Sup.	Fix. Space	Fix. Time
Trajectory	Feuerhake and Sester (2013)		X	X		
	Chen et al. (2014)		X		X	X
	Li and Fu (2014)	X		X		
	Aydin and Angryk (2016)	X			X	X
	Huang et al. (2016)		X		X	X
	Lee et al. (2016)	X		X		
	Ibrahim and Shafiq (2019)		X	X		
	Aydin et al. (2020)	X			X	X
	He et al. (2020)	X		X		
Fixed Position	Tsoukatos and Gunopulos (2001)	X		X		
	Julea et al. (2008)	X		X		
	Julea et al. (2011)	X			X	
	Leong and Chan (2012)	X		X		
	Flamand et al. (2014)	X			X	X
	Gurram and Rama Mohan Reddy (2014)	X			X	X
	Alatrista-Salas et al. (2015)		X		X	
	Chen et al. (2015)	X			X	X
	Sunitha and Rama Mohan Reddy (2016)	X			X	X
	Alatrista-Salas et al. (2016)		X		X	
	Xue et al. (2016)	X		X		
	Yusof and Zurita-Milla (2017)		X	X		
	Chen et al. (2017)	X			X	X
	Batu et al. (2017)	X		X		
	Campisano et al. (2018)	X				
	Zhang et al. (2018)	X		X		
	Chen et al. (2020)		X			X
	Koseoglu et al. (2020)	X			X	X
	Ben Chaabene et al. (2021)	X		X		

As can be seen, our previous work (Campisano et al., 2018) is the only approach that is similar to this article. It searches for frequent sequences in time intervals and regions in space, starting to group sequences in space and, then, in time. This way, it considered space linearly. The present article extends the previous one with a new perspective on the problem. First, we group sequences in time and then in space. It enables generalization to support three-dimensional space.

To the best of our knowledge, the approach proposed in this article is the first able to find sequences constrained in time and three-dimensional space in an integrated manner. Thus, for comparison, we also implemented an intuitive approach of applying SPADE with low support followed by DBSCAN, which guarantees the desired constraints related to time and space. This intuitive approach (SPADE + DBSCAN) is based on Alatrista-Salas et al. (2015). This work grouped data according to distance and extracts sequential patterns to take into account the temporal aspect. Based on this, SPADE + DBSCAN combines an algorithm for sequence mining with a clustering method to find clusters that contain frequent sequences in three steps: i) find all the frequent sequences, using SPADE (Sequential PAttern Discovery using Equivalence classes) algorithm that discovers sequential patterns and minimizes I/O and computational costs (Zaki, 2001); ii) for each frequent sequence find all its occurrences over the dataset; and iii) cluster the occurrences of each frequent sequence, using DBSCAN (Density-Based Clustering of Applications with Noise) clustering algorithm based on the notion of density (Ester et al., 1996). Thus, it is possible to discover sequences and clusters them constrained in space and time.

3 Generalized Spatial-Temporal Sequence Mining

This section presents the GSTSM. Section 3.1 provides background definitions, while 3.2 presents spatial-temporal ranged-group principles. Section 3.3 presents the main algorithm of GSTSM. Section 3.4 presents how to obtain kernel range groups, whereas Section 3.5 describes how to merge kernel range groups to obtain solid range groups. Later, Section 3.6 details how to generate sequence candidates. Finally, Section 3.7 provides an example using synthetic data to explain all concepts.

3.1 Basic definitions

A **Time-stamped Sequence (TS)** t is an ordered sequence of items in time: $t = <v_1, v_2, \cdots, v_n>$, where v_i is an item, $|t| = n$ is the number of items in t, v_n is the most recent item in t (Shumway and Stoffer, 2017). A **subsequence** is a continuous sample of a TS with a defined length. The p-th subsequence of size m in a TS t, represented as $sub_{m,p}(t)$, is an ordered sequence of items $<v_p, v_{p+1}, \cdots, v_{p+m-1}>$, where $|sub_{m,p}(t)| = m$ and $1 \leq p \leq |t| - m$. A **sequence** $s = <w_1, w_2, \cdots, w_k>$ is **included** in a TS t, if exists a starting position q, such that $w_1 = v_q, w_2 = v_{q+1}, \cdots, w_k = v_{q+k-1}$.

A **position** p is given by an ordered triple (x, y, z), where x, y and z are the coordinates of one point in the Cartesian coordinate system. Let f and

h be two positions, where $f = (x_f, y_f, z_f)$ and $h = (x_h, y_h, z_h)$. The distance between f and h, denoted by $dist(f,h)$, is the euclidean distance between two points, *i. e.*, $dist(f,h) = \sqrt{(x_h - x_f)^2 + (y_h - y_f)^2 + (z_h - z_f)^2}$.

Let $P = \{p_1, p_2, ..., p_m\}$ be a set of positions, a **STS** st is a couple (p,t) where $p \in P$ is a position, and t is the associated TS. A **STS dataset** D is a set of STS. An STS $st = (p,t)$ is said to support a sequence s, if s is a subsequence in t: $sup(s, st) = |Q|, \forall q \in Q \mid s = sub_{|s|,q}(st.t)$. The frequency of a sequence s in st is the fraction of $st.t$ that supports s: $freq(s, st) = \frac{sup(s,st)}{|st.t|}$.

The **support** of a sequence s in D is the number of timestamps in D in which s is **included**: $sup(s, D) = |Q|, \forall q \in Q, \exists st_i \in D \mid s = sub_{|s|,q}(st_i.t)$, where Q is the set of sequence time stamps s in D. The **frequency** of a sequence s in D is the fraction of time in D that supports s: $freq(s, D) = \frac{sup(s,D)}{|st.t|}$, $st \in D$, assuming that $|st.t|$ is the same across all TS. Given a user's minimum threshold $\gamma \in]0..1]$, a sequence s is said to be **frequent** if $freq(s, D) \geq \gamma$.

A **time range** (or simply **range**) $r = (r_s, r_e)$ is defined by a start time r_s and an end time r_e. The length of a time range r is given by $|r| = r_e - r_s + 1$. We define the set of all potential time ranges over D as PR.

3.2 Spatial-temporal ranged-group principles

To address the defined problem, we introduce spatial-temporal ranged-groups: Ranged-Group, Kernel Ranged-Group, and Solid Ranged-Group. These concepts were inspired by temporal ranges proposed by Saleh and Masseglia (2008). These concepts are fundamental in this paper and are presented below.

A **spatial group** (or simply **group**) g is defined by a set of spatial positions that are at maximum distance σ from at least one other position of this group, *i.e.*, $g \mid \forall p \in g, \exists q \in g | dist(p,q) \leq \sigma$, where σ is user's threshold. We define the set of all potential groups of positions over D as PG. The set of STS of a group g is defined by $sts(g) = SG \mid \forall st \in SG, st.p \in g$.

A **Ranged-Group (RG)** rg is a triple (s, r, g), where s is a sequence, r is a time range, and g is a group. The **occurrences** of a *sequence* s in a RG rg, defined by $occur(s, r, g)$, refer to the number of all occurrences of s in range r on $sts(g)$. The **support** of a *sequence* s in a RG rg, $sup(s, r, g)$ is the number of timestamps that s starts in the range r on $sts(g)$, *i.e.*, $sup(s, r, g) = |Q|, \forall q \in Q, \exists st \in sts(g) \mid s = sub_{|s|,q}(st.t), r_s \leq q \leq r_e, |s| \leq r_e$. The **frequency** of a *sequence* s in a RG rg, $freq(s, r, g)$ is the fraction of support of the RG $sup(s, r, g)$ over the size of r: $freq(s, r, g) = \frac{sup(s,r,g)}{|r|}$.

Given γ and β, the user's minimum thresholds for frequency and size of the group, respectively, we introduce the characteristics of Kernel Ranged-Group (KRG) in Definition 1 and Solid Ranged-Group (SRG) in Definition 2.

Definition 1 *Let rg be a RG with sequence s, time range r, and group g. Then, rg is called a* **KRG** *if and only if the following conditions hold:*

1) $freq(s, r, g) \geq \gamma$
2) $|g| \geq \beta$
3) $\forall r' \in PR \mid r' \subset r$ *and* $r'_s = r_s$, *we have both:*
a) $freq(s, r', g) \geq \gamma$
b) $sup(s, r', g) < sup(s, r, g)$
4) $\forall g' \in PG \mid g \subseteq g', occur(s, r, g') = occur(s, r, g)$
5) $\forall g' \in PG \mid g' \subset g, occur(s, r, g') < occur(s, r, g)$

The first condition in Definition 1 ensures that the sequence s of rg is frequent in STS of an associated group (*i.e.*, $sts(g)$) over time range r. The second condition ensures that the group g has a minimum size defined by the user. The third condition ensures that the size of r is minimal among ranges that start at the same timestamp. If a smaller range exists starting at the same timestamp, then, in this range, s is frequent on $sts(g)$, but the support of s is decreased. The fourth condition ensures that the size of g is maximal. So if a larger group exists that contains g where s is frequent in the range r, the number of occurrences s on STS of this group is the same. It is not worth increasing the group from g to g' since the raise does not contribute to the number of occurrences of s. Finally, the fifth condition ensures that the size of g is minimal. Suppose a smaller group exists where s is frequent in the range r. In that case, the number of occurrences s on STS of this group decreases. It is not worth decreasing the group from g to g' since it reduces the number of occurrences of s.

Definition 2 *Let rg be a RG with sequence s, time range r, and group g. Then, rg is called a* **SRG** *if and only if the following conditions hold:*
1) $freq(s, r, g) \geq \gamma$
2) $|g| \geq \beta$
3) $\forall r' \in PR \mid r \subseteq r'$, *we have either a) or b) or both:*
a) $freq(s, r', g) < \gamma$
b) $sup(s, r', g) = sup(s, r, g)$
4) $\forall r' \in PR \mid r' \subset r, sup(s, r', g) < sup(s, r, g)$
5) $\forall g' \in PG \mid g \subseteq g', occur(s, r, g') = occur(s, r, g)$
6) $\forall g' \in PG \mid g' \subset g, occur(s, r, g') < occur(s, r, g)$

The first condition in Definition 2 ensures that the sequence s of rg is frequent in $sts(g)$ over time range r. The second condition ensures that the group has a minimum size defined by the user. The third condition ensures that the size of r is maximal. If a larger range exists, s is not frequent on $sts(g)$, or the support of s is the same (*i.e.*, it is not worth extending the range from r to r', since the extension is not going to contribute to the support of s on $sts(g)$). The fourth condition ensures that the size of r is minimal. The sequence s is supported by the first and last time in r. Suppose a smaller range exists where s is frequent. In that case, the support is going to be lower (*i.e.*, relevant STS supporting s would have been dropped from the range and should be kept). The fifth condition ensures that the size of g is maximal. So if a larger group exists that contains g where s is frequent in the range r, the number of occurrences s on STS of this group is the same. It is not worth increasing the group from g to g' since the raise does not contribute to the number of occurrences of s). Finally, the sixth condition ensures that the size of g is minimal. Suppose a smaller group exists where s is frequent in the range r. In that case, the number of occurrences s on STS of this group is the smaller (*i.e.*, it is not worth decreasing the group from g to g', as this also reduces the number of occurrences of s).

Let k be the size of s, then srg is a k-SRG. $\mathbf{SRG_k}$ is the set of all k-SRG.

3.3 G-STSM Algorithm

This section presents the G-STSM. Our algorithm is designed to the identification of frequent sequences in STS datasets from the concept of SRG. The notion of ranged-group (RG, KRG, and SRG) introduced in the previous section enables for extracting SRG efficiently.

The G-STSM is based on the candidate-generating principle. Our goal is to start finding SRGs for sequences of size one. Then we explore the support and the number of occurrences of SRGs for larger sequences with a limited number of scans over the database. To this end, we need to find the range and the set of positions (*i. e.*, the SRG) in which a candidate sequence is frequent in only one scan.

Let $c \in C_k$ be a candidate of size k in the set of candidates (C_k). Then, in our data structure, c is associated with $c.seq$, the sequence of items, $c.time$, the range of possible frequency, $c.pos$, the set of spatial positions, and $c.rgs$, the set of RG of $c.seq$ over $c.time$ with respect to γ in $c.pos$.

A RG rg is associated with a sequence $rg.s$, a range $rg.r$, and a group of positions $rg.g$. In addition, $rg.freq$ is the frequency of the sequence $rg.s$ in rg, and $rg.occ$ is a set of occurrences of $rg.s$ in rg. Finally, a logical value ($rg.closed$) lets you know the status of the RG. When it is true means that, at some point, while scanning the dataset to stretch its range, its frequency has become less than the minimum user-defined frequency γ. A KRG krg has all the information of a RG, where it respects the user defined thresholds γ, β and σ. Likewise, an SRG is associated with the same information as a RG.

Algorithm 1 aims to generate SRGs from the size of sequences 1 to k. It receives as input an STS dataset D, a set of items I, a set of spatial positions P, and user's thresholds, γ, β and σ. It has three main functions for (i) finding the KRGs (detailed in Section 3.4), (ii) merging the KRGs for the identification of SRGs (detailed in Section 3.5), and (iii) generating the candidates (detailed in Section 3.6).

The algorithm starts from candidates of size 1 sequences. They are built from all distinct items of I presented in D, considering its entire range and all spatial positions P (lines 3-5). Then, a repeat-until loop (lines 6-25) computes all SRGs of size k (SRG_k). For this, a scan over the dataset D is first performed

Algorithm 1: G-STSM

```
   Input: $D, I, P, \gamma, \beta, \sigma$
1  begin
2      $k \leftarrow 0$
3      foreach ($i \in I$) do
4          $C_1 \leftarrow C_1 + (i, [D_s..D_e], [P_1..P_n], \emptyset)$
5      end
6      repeat
7          $k++$
8          $SRG_k \leftarrow \emptyset$
9          $SW \leftarrow slidingWindows([D_s..D_e], k)$
10         foreach ($d \in SW$) do
11             foreach ($c \in C_k / d.timestamp \in c.time$) do
12                 $c.rgs \leftarrow findKRGs(c, d, \gamma, \beta, \sigma)$
13             end
14         end
15         foreach ($c \in C_k$) do
16             $c.rgs \leftarrow validateGroup(c.rgs, \beta)$
17         end
18         foreach ($c \in C_k$) do
19             $c.rgs \leftarrow mergeKRGs(c.rgs, \gamma)$
20             foreach ($rg \in c.rgs$) do
21                 $SRG_k \leftarrow SRG_k + (rg)$
22             end
23         end
24         $C_{k+1} \leftarrow genCandidates(SRG_k)$
25     until ($C_{k+1} \neq \emptyset$);
26 end
   Output: $SRG$
```

(lines 10-14) from the set of transactions d using a sliding window SW of length k (line 9). As it is looking for sequences of size k, it is necessary to deal with k transactions at time, the number of transactions is determined by the size of the sequence (k) to be found. The goal is to find the KRGs of each candidate $c \in C_k$ in the sliding window (line 12).

After executing the $findKRGs$ algorithm, some RGs may not have been closed (*rg.closed*=*True*). This is because at the end of the dataset scan, the algorithm is still trying to stretch some RG and as their frequency remains satisfying γ, they are not validated for the size of their respective groups. Thus, the *validateGroup* function is executed (line 16) for each candidate to solve this problem, setting its *rg.closed* attribute to *True* and checking if its group size is equal to or greater than β (lines 15-17). Next, for each candidate (lines 18-23), we merge these found KRGs (line 19), and from the merged KRGs, we identify the SRGs SRG_k (lines 20-22). Finally, candidate sequences of size $k+1$ are computed from SRG_k (line 24). Once we get empty candidates sequences of size $k+1$, the loop stops (line 25).

The *validateGroup* function is shown in Algorithm 2. It receives as input the set of RGs (RG) from a candidate and the minimum size of a group (β). It starts defining a set of elements that will be removed (*toRemove*) from the set of RGs (line 2), if it does not have the minimum group size. A loop start (lines 3-8) iterating over each RG that has the *rg.closed* attribute set to *False*, setting it to *True* (line 4) and checking the length of the group (line 5), if it does not respect the minimum group size the RGs is added to the *toRemove* set, and removed from the set of RGs (lines 9-11). As output, the algorithm provides the updated set of RG received, all of which are now validated as to the minimum size of the group. Thus, a set of KRG.

Algorithm 2: validateGroup

```
   Input: $RG, \beta$
1  begin
2      $toRemove \leftarrow \emptyset$
3      foreach ($rg \in RG | rg.closed = False$) do
4          $rg.closed \leftarrow True$
5          if ($length(rg.g) < \beta$) then
6              $toRemove \leftarrow toRemove + rg$
7          end
8      end
9      foreach ($k \in toRemove$) do
10         $RG \leftarrow RG - k$
11     end
12 end
   Output: $RG$
```

3.4 Discovery of Kernel Ranged-Groups

The goal of Algorithm 3 is to find the KRG information for a candidate c. It receives as input a candidate c, the set of transactions d from a sliding window of SW, and the thresholds defined by the user (γ, β, and σ). First, the algorithm invokes the *SplitGroups* function (line 3) to find the positions where the candidate sequence occurs in the current set of transactions d. It generates groups based on the group definition that respects the threshold σ and the positions of the occurrences. For each group, the algorithm creates a new RG having the range in which start and end are equals to the start of the current set of transactions ($krg.r = [d.start, d.start]$), and its respective group. These new RGs are added to the set of an existing one.

From all RGs computed, including those that already existed and those created on line 3, the algorithm starts trying to stretch them all. For each two different RGs q and r, belonging to a candidate's RG set $c.rgs$, which are opened, the algorithm tries to perform the stretch (lines 8-24). It can occur as long as positions of both RGs produce a single group (line 11). Besides, such new RG should also have a frequency greater than or equal to the minimum γ (line 18). If all conditions are met, the new RG is added to the RGs set of the candidate sequence $c.rgs$ (line 19). The RGs that generated the new one are added to the set *toRemove* of elements that will be removed (line 21). All elements stretched from the $c.rgs$ set are removed (lines 25-27). Finally, the

Algorithm 3: findKRGs

```
Input: $c, d, \gamma, \beta, \sigma$
1  begin
2      if ($c.seq \subseteq d$) then
3          $c.rgs \leftarrow c.rgs + SplitGroups(c, d, \sigma)$
4          $groupable \leftarrow True$
5          while ($groupable$) do
6              $groupable \leftarrow False$
7              $toRemove \leftarrow \emptyset$
8              foreach ($q, r \in c.rgs \mid q.closed = False \wedge r.closed = False \wedge r \neq q$) do
9                  $pos \leftarrow q.group \cup r.group$
10                 $group \leftarrow CreateGroup(pos, \sigma)$
11                 if ($pos - group = \emptyset$) then
12                     $new\_krg.r_s \leftarrow min(q.r_s, r.r_s)$
13                     $new\_krg.r_e \leftarrow max(q.r_e, r.r_e)$
14                     $new\_krg.g \leftarrow group$
15                     $new\_krg.occ \leftarrow q.occ \cup r.occ$
16                     $new\_krg.freq \leftarrow \frac{suport(new\_krg.occ)}{|[new\_krg.r_s..new\_krg.r_e]|}$
17                     $new\_krg.closed \leftarrow False$
18                     if ($new\_krg.freq \geq \gamma$) then
19                         $c.rgs \leftarrow c.rgs + new\_krg$
20                         $groupable \leftarrow True$
21                         $toRemove \leftarrow toRemove + q + r$
22                     end
23                 end
24             end
25             foreach ($k \in toRemove$) do
26                 $c.rgs \leftarrow c.rgs - k$
27             end
28             $c.rgs \leftarrow validateFrequencyAndGroup(c.rgs, timestamp, \gamma, \beta)$
29         end
30     end
31 end
Output: Kernel Ranged-Group(s) of $c$ updated
```

RGs are validated using the *validateFrequencyAndGroup* function (line 28). As a result, the algorithm updates the set of RGs of the received candidate c. Thus, after each step over the dataset, Algorithm 3 generates all possible KRGs.

The *validateFrequencyAndGroup* function is presented in Algorithm 4. Its objective is to verify that the user thresholds were observed in each RGs, checking if they can still be stretched by keeping the frequency greater than or equal to the minimum γ and if the minimum group size β occurs. It takes as input a set of RGs *RG* of a candidate sequence, the timestamp of the start of the current sliding window *timestamp*, the user-defined thresholds γ and β.

For each element of the set *RG* with attribute *rg.closed* with value *False* ($krg.closed = False$) (line 3 - 11), the frequency is calculated by stretching the range of the RG to the start of the sliding window (*timestamp*) (line 4). Afterwards, it is checked if the frequency is less than the threshold γ (lines 5 -

9). If this happens, the state (*rg.closed*) is changed to *True* (line 6), meaning that the RG cannot be stretched anymore, and it is verified if the group size was below the threshold β, marking such element for removal if this occurs (lines 7 - 9). Finally, all marked elements are removed from the *RG* set (lines 12 - 14). As output, the function returns the updated *RG* set.

Algorithm 4: validateFrequencyAndGroup

```
   Input: $RG, timestamp, \gamma, \beta$
1  begin
2      $toRemove \leftarrow \emptyset$
3      foreach $(rg \in RG | rg.closed = False)$ do
4          $freq \leftarrow \frac{support(rg.occ)}{|[rg.r_s..timestamp]|}$
5          if $(freq < \gamma)$ then
6              $rg.closed \leftarrow True$
7              if $(length(rg.g) < \beta)$ then
8                  $toRemove \leftarrow toRemove + rg$
9              end
10         end
11     end
12     foreach $(k \in toRemove)$ do
13         $RG \leftarrow RG - k$
14     end
15 end
   Output: $RG$
```

Lemma for find Kernel Ranged-Groups: Let c and d be a candidate and the current set of transactions of D, respectively. Algorithm 3 together with *validateGroup* function makes it possible to find all the KRGs.

Proof: Let $rg \in c.rgs$, be a KRG of the sequence s over range r and group g after *validateGroup* function (*i.e.* rg cannot be discovered through stretch with any other KRG of status $closed = False$ in $c.rgs$), then:

1. $freq(s, r, g) \geq \gamma$. According to KRG definition (Definition 1), s is frequent in each KRG. Thus, Algorithm 3 generates the initial RGs which s is frequent and, if rg is the new RG resulting from a stretch process, it checks the frequency of s on rg. Suppose $freq(s, r, g) < \gamma$, Algorithm 3 could not have created rg as a KRG, which would be a contradiction.
2. $|g| \geq \beta$. According to Definition 1, g has a minimum size. Thus, Algorithm 3 or *validateGroup* function validated the rg checking the user's threshold β. Suppose $|g| < \beta$, Algorithm 3 or *validateGroup* could not have validated the size of g and rg would not be KRG, which would be a contradiction.
3. $\forall a$ such that $a \subset r$ and $a_s = r_s$, we have both cases:
   – s is supported by the first and last timestamps in r. Then, s has lower support on any sub-range of r.
   – $freq(s, a, g) \geq \gamma$, otherwise it would not have been stretched to achieve r, which would be a contradiction.

4. $\forall b$ in a group such that $g \subseteq b$, we have by definition that $occur(s, r, b-g) = 0$, then $occur(s, r, b) = occur(s, r, g)$.
5. s occurs in all $sts(g)$, otherwise if exists a TS which does not contain an occurrence of s, it would not be included in $sts(g)$. Then, the occurrence of any subset b of g is smaller than in g: $occur(s, r, b) < occur(s, r, g)$.

Based on the five observations above, $c.rgs$ is formed by all discovered KRGs of s on PR and PG concerning γ, β, and σ. Thus, KRG is the set of all KRGs $krg = (s, r, g)$ on PR and PG concerning γ, β, and σ, that respect Definition 1. The KGRs are then used to form the SRGs through the merge process described below.

3.5 Merge of Kernel Ranged-Groups

When searching for occurrences of a candidate sequence over the dataset, the G-STSM steps over each timestamp trying to stretch the RGs. Sometimes, without an occurrence in a timestamp, it is not possible to stretch a RG, as its frequency falls below the minimum γ. As a result, no attempt is made to stretch the RG. However, the same sequence may occur in a later set of timestamps, and, at this moment, it would be possible to stretch that RG. However, a new one was already generated in the later step. To solve this issue, Algorithm 5 tries to merge KRGs with a larger range and possibly a larger group.

The goal of the Algorithm 5 is to merge KRGs. Let q and u be two different KRGs from the same candidate sequence. They can be merged into a group $qu = q \cup u$ as long as they have an intersection (line 7) and qu has a frequency greater than or equal to the minimum frequency defined by the user (line 10).

The Algorithm 5 receives as input a set of KRG from a candidate sequence KRG and the minimum frequency defined by the user γ. The algorithm starts by defining a flag *mergeable* as *True* (line 2), while it is *True*, it tries to join KRGs (lines 3 - 20). The flag is defined as *False* (line 4), and a set *toRemove* of elements that will be removed from the set KRG once they have been merged, generating a new KRG is created (line 5). Then, each pair of elements in KRG (lines 6 - 16) is checked to know if they have an intersection in the space (line 7) and if the union of the elements maintains a frequency equal or greater than the minimum threshold defined by the user (line 10). The elements are merged and added to the KRG set (line 11). The flag *mergeable* is, therefore, updated to *True* (line 12) and the two merged elements are added to the *toRemove* set. Finally, all elements that have been merged from the KRG set are removed (lines 17-19). As a result, the algorithm returns the input set KRG containing the merged KRGs, when possible. Thus, when the execution of Algorithm 5 ends, it has joined KRGs respecting the conditions of Definition 2, so we have a set of SRG.

Lemma for merge Kernel Ranged-Groups: Let KRG be the set of KRG of s over range r and group g concerning the thresholds γ, β, and σ. Algorithm 5 makes it possible to find all the SRGs.

Algorithm 5: mergeKRGs

```
   Input: $KRG, \gamma$
1  begin
2      $mergeable \leftarrow True$
3      while ($mergeable$) do
4          $mergeable \leftarrow False$
5          $toRemove \leftarrow \emptyset$
6          foreach ($q, u \in KRG \mid q \neq u$) do
7              if ($(q.g \cap u.g) \neq \emptyset$) then
8                  $start \leftarrow min(q.r_s, u.r_s)$
9                  $end \leftarrow max(q.r_e, u.r_e)$
10                 if ($\frac{support(q.occ \cup u.occ)}{|[start..end]|} \geq \gamma$) then
11                     $KRG \leftarrow KRG + q \cup u$
12                     $mergeable \leftarrow True$
13                     $toRemove \leftarrow toRemove + q + u$
14                 end
15             end
16         end
17         foreach ($k \in toRemove$) do
18             $KRG \leftarrow KRG - k$
19         end
20     end
21 end
   Output: KRG
```

Proof Let $krg \in KRG$, be a SRG of s on r and g after Algorithm 5 (*i.e.* krg cannot be merged with any other KRG in KRG), then:

1. $freq(s, r, g) \geq \gamma$. According to SRG definition (Definition 2), s is frequent in each SRG. Thus, Algorithm 5 generates KRGs which s is frequent and, if krg is the new KRG resulting from a merge process, it checks the frequency of s on krg or krg is not a result from a merge process, so it does not go through a validation process, but the frequency has been checked using Algorithm 3. Suppose $freq(s, r, g) < \gamma$, Algorithm 5 could not have created krg as a SRG, which would be a contradiction.
2. $|g| \geq \beta$. According to Definition 2, g has a minimum size. Thus, if krg is the result of a merging process, then the group g is formed by Algorithm 5 with the union between the positions of merged RGs (that also satisfy the minimum size condition of groups by Definition 1); or krg is not a result from a merge process but the size of the group have been checked using Algorithm 3. Suppose $|g| < \beta$, Algorithm 5 could not have generated this group by a merge process or Algorithm 3 and *validateGroup* function could not validate krg as a KRG, which would be a contradiction.
3. $\forall a$ such that $r \subseteq a$, we have one of the following cases:
   – $sup(s, a - r, g) > 0$, then s is not frequent in a (otherwise, let us consider r' the range to which s is present in a, then r and r' would have been merged).
   – $sup(s, a - r, g) = 0$, then $sup(s, a, g) = sup(s, r, g)$ (in this case, s may remain frequent in a or not, depending on the size of a).

4. According to Definition 2, s is supported by the first and last timestamps in r. Then, s has lower support on any sub-range of r.
5. $\forall b$ in a group such that $g \subseteq b$, we have by definition that $occur(s, r, b-g) = 0$, then $occur(s, r, b) = occur(s, r, g)$.
6. According to Definition 2, s occurs in all $sts(g)$, otherwise if exists a TS in $sts(g)$ which does not contain an occurrence, it would not be included in g. Then, the occurrence of any subset b of g is smaller than in g: $occur(s, r, b) < occur(s, r, g)$.

Based on the six observations above, KRG is updated with the set of RGs resulting of the merge process of all KRGs of s on PR and PG concerning γ, β, and σ. Thus, SRG is the set of all SRGs $srg = (s, r, g)$ on PR and PG concerning γ, β, and σ discovered from the KRG previously computed. Finally, we have all the SRGs of size k, which are the sequences restricted in space and time that respect Definition 2, reaching our goal.

3.6 Generation of candidates

The Algorithm 6 combines SRGs that have sequences of size k, received as input, to generate candidates with sequences of size $k+1$. Let x and y be SRGs, the conditions for this to occur are (line 3): that we have an intersection of candidates over the time range ($[x.r_s..x.r_e] \cap [y.r_s..y.r_e]$), intersection over the set of spatial positions ($x.g \cap y.g$), and a common subsequence: $<x.s_2, \ldots, x.s_k>=<y.s_1, \ldots, y.s_{k-1}>$.

Algorithm 6: genCandidates

```
  Input: $SRG_k$
1 begin
2   $C_{k+1} \leftarrow \emptyset$
3   foreach $(x, y \in SRG_k)$ such that: $(((x.s_2, \ldots, x.s_k) = (y.s_1, \ldots, y.s_{k-1})) \wedge ([x.r_s..x.r_e] \cap [y.r_s..y.r_e] \neq \emptyset) \wedge ((x.g \cap y.g) \neq \emptyset))$ do
4     $z \leftarrow (x.s_1, \ldots, x.s_k, y.s_k)$
5     $p \leftarrow [x.r_s \ldots (y.r_e - 1)]$
6     $u \leftarrow (x.g \cap y.g)$
7     $C_{k+1} \leftarrow C_{k+1} + (z, p, u, \emptyset)$
8   end
9 end
  Output: $C_{k+1}$ set of candidates having length $k+1$
```

If all restrictions are respected, a candidate is defined with a new sequence (line 4), a new range (line 5), and a new group of spatial positions (line 6). The time range referring to the candidate is generated from the start time r_s of the first sequence and the end of the second sequence minus one $r_e - 1$. Besides, the set of positions is equal to the intersection of the SRGs that generate the candidate. Finally, the new candidate is added to the set of candidates C_{k+1}

(line 7). As a result, the function returns a set of candidate sequences of size $k+1$.

The antimonotonic concept is present when checking a common subsequence of both SRGs. For example, the existence of sequences of size two AB and BC enables the generation of a candidate of size three ABC (Mooney and Roddick, 2013).

3.7 Synthetic Data Example

This section provides an example to better illustrate how the G-STSM algorithm works. A synthetic STS dataset D is shown in Figure 1.a. It presents nine STS, each one associated with a TS $<t_{11}, t_{12}, t_{13}, t_{21}, t_{22}, t_{23}, t_{31}, t_{32}, t_{33}>$, represented by the columns of the table, and a spatial position p, where $p \in P \mid P = \{p_{11}, p_{12}, p_{13}, p_{21}, p_{22}, p_{23}, p_{31}, p_{32}, p_{33}\}$. Each TS has three observations. The distribution of positions can be seen in Figure 1.b, which shows two orthogonal axes and the distribution of points in relation to these axes. All positions are in the same plane. The unit of measurement is not important for the algorithm, but it is necessary that the distribution of positions and σ use the same scale.

	t_{11}	t_{12}	t_{13}	t_{21}	t_{22}	t_{23}	t_{31}	t_{32}	t_{33}
v_1	d	a	c	-	a	c	b	b	b
v_2	d	d	-	a	b	a	a	-	a
v_3	b	b	b	-	-	c	-	-	c

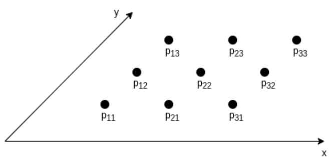

Fig. 1 Synthetic STS dataset D. Nine STS, each one with three observations. Some cells have a dash (-) representing no value (a). Distribution of positions of the STS (b).

The input data for the algorithm are the STS dataset D, the set of distinct items I in D, the set of positions P, and the user's thresholds: $\gamma = 60\%$, $\beta = 2$, and $\sigma = 1$.

For the sake of simplicity, we show only the most important attributes. The process is based on the principle of the generation of candidates. It starts with a candidate for each item in I. Then, for each candidate, we seek for occurrences through each timestamp, generating KRGs of sequences of size one. After executing algorithm *findKRGs* and *validateGroup*, we have five KRGs:

i $(a, [1,2], \{p_{12}, p_{21}, p_{22}, p_{23}, p_{31}, p_{33}\})$
ii $(b, [1,3], \{p_{11}, p_{12}, p_{13}, p_{22}, p_{31}, p_{32}, p_{33}\})$
iii $(c, [1,1], \{p_{13}, p_{23}\})$
iv $(c, [3,3], \{p_{23}, p_{33}\})$
v $(d, [1,2], \{p_{11}, p_{12}\})$

It is worth mentioning that there are two KRGs referring to sequence c. They appear separate because after reading the first timestamp, sequence c

does not occur in the second timestamp, reducing the RG frequency to 50% and updating its attribute *rg.closed* to *True*. Then, an occurrence in the last timestamp creates a new RG, but the first one has *rg.closed* attribute set to *True* and cannot be stretched.

Another point to comment on is candidate d, which has a group of only one at timestamp 1. It is because group validation size only occurs when the RG cannot be stretched anymore. As it occurs in the second timestamp (with a group size 2) validated only by *validateGroup* function, and so kept.

Finally, it is important to see that three RG of sequence a are merged in only one. One from the first timestamps and two from the second timestamp. Although they do not have an intersection of their positions, they form a single group using all the positions.

After running $mergeKRGs$, we have four SRGs, because KRGs of sequence c where merged:

i $(a, [1, 2], \{p_{12}, p_{21}, p_{22}, p_{23}, p_{31}, p_{33}\})$
ii $(b, [1, 3], \{p_{11}, p_{12}, p_{13}, p_{22}, p_{31}, p_{32}, p_{33}\})$
iii $(c, [1, 3], \{p_{13}, p_{23}, p_{33}\})$ and
iv $(d, [1, 2], \{p_{11}, p_{12}\})$

They are used to generate candidates of size two:

i $(aa, [1, 1], \{p_{12}, p_{21}, p_{22}, p_{23}, p_{31}, p_{33}\})$
ii $(ab, [1, 2], \{p_{12}, p_{22}, p_{31}, p_{33}\})$
iii $(ac, [1, 2], \{p_{23}, p_{33}\})$
iv $(ad, [1, 1], \{p_{12}\})$
v $(ba, [1, 1], \{p_{12}, p_{22}, p_{31}, p_{33}\})$
vi $(bb, [1, 2], \{p_{11}, p_{12}, p_{13}, p_{22}, p_{31}, p_{32}, p_{33}\})$
vii $(bc, [1, 2], \{p_{13}, p_{33}\})$
viii $(bd, [1, 1], \{p_{11}, p_{12}\})$
ix $(ca, [1, 1], \{p_{23}, p_{33}\})$
x $(cb, [1, 2], \{p_{13}, p_{33}\})$
xi $(cc, [1, 2], \{p_{13}, p_{23}, p_{33}\})$
xii $(da, [1, 1], \{p_{12}\})$
xiii $(db, [1, 2], \{p_{11}, p_{12}\})$
xiv $(dd, [1, 1], \{p_{11}, p_{12}\})$

Then, after executing $findKRGs$ and *validateGroup* for the sequence of size two, we obtain:

i $(ac, [2, 2], \{p_{23}, p_{33}\})$; and
ii $(db, [1, 2], \{p_{11}, p_{12}\})$.

Note that sequences ab, ad, ba, ca, and dd are found, but they are discarded due to the size of their groups.

Since no merge happens when running $mergeKRGs$, they are also SRGs. Besides, since we do not have any candidate sequences of size three, the algorithm finishes.

It is worth mentioning that even if this small dataset were part of a large dataset with thousands of records, the discovered patterns would be found in the same way.

4 Experimental Evaluation

This paper focuses on identifying frequent sequences constrained in space and time. An intuitive approach to solve this problem is applying SPADE with DBSCAN. Section 4.1 is driven to compare the performance of G-STSM to SPADE + DBSCAN. Besides, Section 4.2 explores the generality of G-STSM using it in the other domain. An additional experimental evaluation discover patterns regarding COVID-19 spread over the state of Rio de Janeiro.

4.1 Seismic Domain

The experiment with the seismic domain focuses on two points. The first is to compare G-STSM and SPADE + DBSCAN approaches regarding both quality and performance. The second is to make a sensitivity analysis of G-STSM parameters, which gives us more information on the algorithm behavior.

The dataset used in the experiments was the Netherlands seismic STS dataset processed using SAX encoding with alphabet size 25 from the F3 Block. The F3 Block dataset corresponds to seismic reflections from the region located in the North Sea's Dutch sector. The seismic data is obtained by sending high-energy sound waves into the ground or seabed as the case. The amplitude of the reflected sound waves is registered. The dataset contains observations related to the time the sound wave arrives and attributes related to the hydrophone position that registered the reflected sound wave, a STS dataset. The results presented in this work were focused on public data of the *inline* T401. Each inline is composed of 951 STSs with 462 observations.

Some evaluations were done using parts of the T401 inline. It was divided into 16 quadrants, as can be seen in Figure 2. The division is shown in straight vertical and horizontal lines in black. Each quadrant has approximately 237 STS and 115 observations. The number of each quadrant is shown in red. The colorful lines represent the previously known seismic horizons. They consist of reflecting interfaces that indicate a stratigraphic boundary between two different regions.Both the dataset and the locations of patterns are available in Lab (2020).

4.1.1 Experimental Setup

As input parameters, G-STSM enables users to set the following thresholds: the minimum frequency (γ), the minimum size of a group (β), and the maximum distance from at least one other position of the same group (σ). The values of γ can vary between 0 and 1. For β, the integer values starting from 2 can be used, and for σ, the integer values starting from 1.

The SPADE algorithm receives an input dataset in a vertical id-list format consisting of a collection of input sequences with events. Each input sequence has a different identifier. Each event in an input sequence also has a unique identifier within that input sequence. Support (*support*) is a parameter that

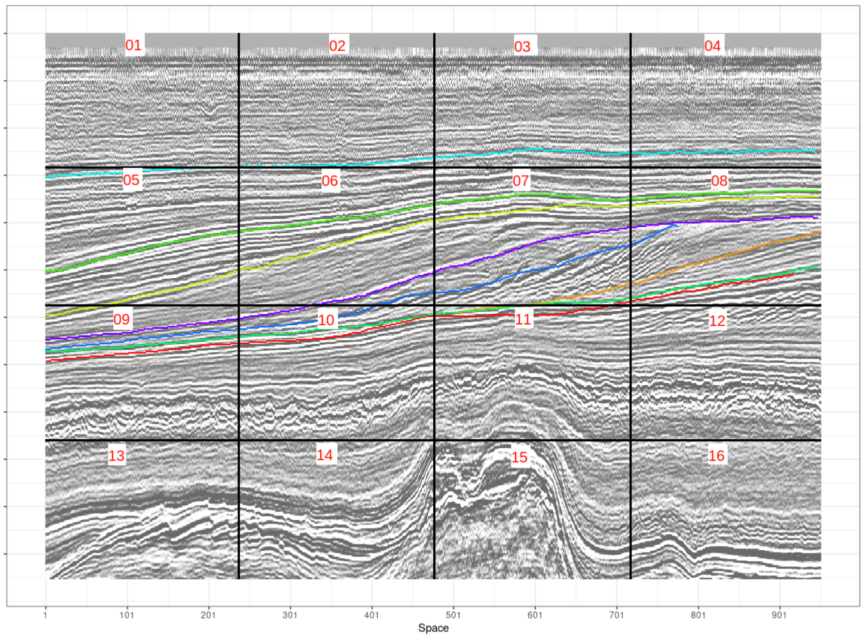

Fig. 2 Dataset, patterns, and division in quadrants. Source: OpendTect (2020).

defines the minimum times a sequence must occur in the dataset to be considered frequent. It was configured dividing β, used in G-STSM, by the number of STSs of the used dataset D, *i.e.*, $support \cong \frac{\beta}{|sts(D)|}$.

DBSCAN receives a matrix with positions of frequent sequences and the input parameters *eps* and *minPts*, and generates the clusters for each frequent sequence. The *eps* parameter is the maximum distance between two points for one to be considered in the neighborhood of the other, and *minPts* parameter is the number of points in a neighborhood to be considered a core point. The epsilon neighborhood *eps* was set to the same value as σ used by G-STSM and the number of minimum points *minPts* in the neighborhood region was configured with the same value as β.

The experiments were conducted on a computer with a Core i7-7820X processor with 16 cores and 128GB of RAM using the Ubuntu 20.04 LTS operation system. The approaches were implemented in R (Team, 2020). We used the *cSPADE* Zaki (2000), an implementation of SPADE algorithm that is available in the *arulesSequences* package (Buchta et al., 2020), and the DBSCAN implementation of the *dbscan* package (Hahsler et al., 2019).

4.1.2 Comparison

Figure 3 shows the total execution time of G-STSM and SPADE + DBSCAN according to the dataset size, starting with a quadrant and doubling the size until using all 16 quadrants (i.e. 1, 2, 4, 8, and 16). G-STSM was set with γ, β, and σ using values 0.6, 10, 5, respectively. SPADE + DBSCAN were set using corresponding values, which were able to discover similar patterns. G-STSM presents the best execution time for all variations. In general, G-STSM is more than two times faster. Note that for the entire dataset, 16 quadrants, G-STSM was even better, being more than three times faster.

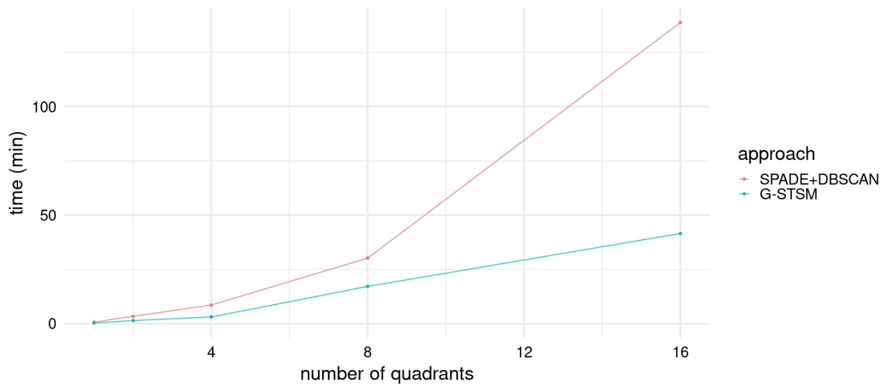

Fig. 3 Total execution time of each approach using a different number of quadrants.

Regarding a qualitative analysis, SPADE + DBSCAN and G-STSM were compared in quadrants 5 to 8 in their ability to match seismic horizons. Whenever an discovered occurrence matches of a horizon, it is computed as true positive. From the results, both approaches obtained similar results, showing difference only from the second decimal place and with a small standard deviation. For the G-STSM a value of 0.96 ± 0.02 was obtained, and 0.95 ± 0.03 for the SPADE + DBSCAN approach.

The results indicates that SPADE + DBSCAN and G-STSM lead to similar discovery, but G-STSM is much more efficient.

4.1.3 Sensitivity Analysis

Figure 4 presents the correlation between the input parameters (γ, β, and σ) and the output information (the number of SRG, the number of occurrences of the sequences contained in the SRG, memory use, and execution time of the G-STSM). G-STSM was set with the values of γ equal to 0.6, 0.8, and 1.0. For β and σ the integer values in the range $[5, 10]$ were applied, using quadrants from 5 to 12.

Note that a lower value of γ (minimum frequency) makes it easier G-STSM to find more occurrences. Hence, it uses more memory because of the extra data and spends more time dealing with it. However, the lower the γ, the fewer SRGs are obtained since SRGs are merged. A smaller β (minimum size of the group) generates more groups and occurrences and spends more memory and time. For larger values of σ (maximum distance from at least one other position of this group), the number of occurrences and SRGs are also higher, the memory usage decreases, and the execution takes longer.

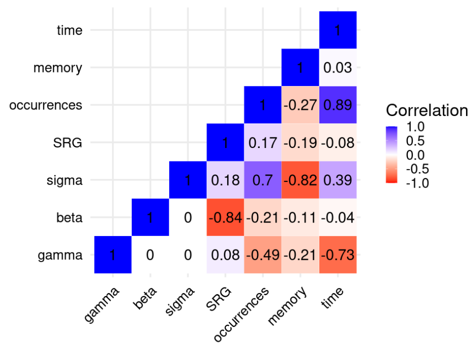

Fig. 4 Correlation between the input parameters (γ, β and σ), resources usage, and the output information.

Figures 5 and 6 show the memory usage and execution time, respectively, with increasing dataset size, starting with a quadrant and doubling the size until using all 16 quadrants (i.e. 1, 2, 4, 8, and 16). For both graphics, it was used three different configurations.

The configuration $HBLS$ (High Beta Low Sigma) is the most restrictive, gets only *denser* SRG, with a greater number of minimum elements in a group ($\beta = 10$) and lower minimum distance of group elements ($\sigma = 5$). Similarly, the configuration $LBHS$ (Low Beta High Sigma) is the least restrictive, enabling few elements in a group ($\beta = 5$) with a high distance from each other ($\sigma = 10$). Finally, $HBHS$ (High Beta High Sigma) has many minimum elements in a group ($\beta = 10$) but enables a large distance from each other ($\sigma = 10$). For all configurations γ was set to 0.8.

As expected, growing up the dataset size, the use of resources gets higher. The greatest time difference occurs with all the datasets (16 quadrants), a difference of 3.9 minutes, 9.82% increase of time from configurations $HBHS$ to $HBLS$. The greatest difference also occurs when using all the datasets for memory usage, 340.6 MB, 10.55% increase in memory use. This behavior indicates that input parameter configurations do not make a great difference in the algorithm performance or resource usage.

4.2 COVID-19 spread

The experiment with the health domain was conducted to evaluate the generality of G-STSM on finding constrained space-time patterns under different domains. G-STSM was used to find Covid-19 spread patterns in a death notification dataset.

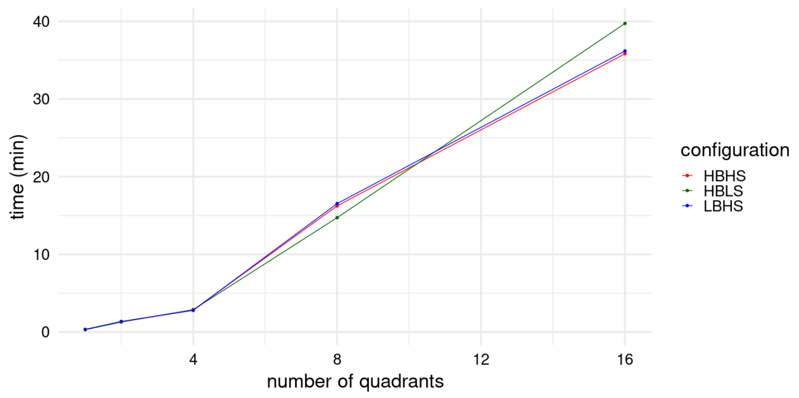

Fig. 5 Time execution of G-STSM using different configurations and dataset sizes.

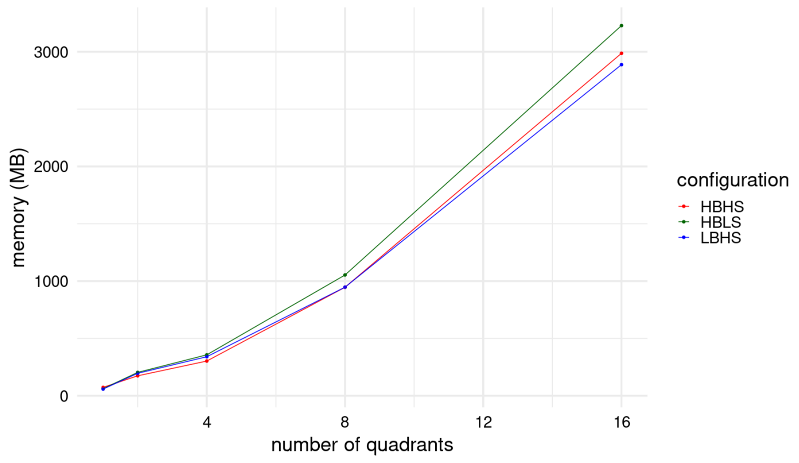

Fig. 6 Maximum memory usage of G-STSM using different configurations and dataset sizes.

The dataset was obtained from Brasil.IO (2021). It compiles daily epidemiological bulletins from the State Health Departments. It is made available in the form of a historical series of confirmed cases and deaths by municipalities. So we have the date and municipality (position) of the occurrences of deaths caused by Covid-19, a STS dataset.

We used the dataset to study the Rio de Janeiro state. It is made up of 92 municipalities, which are organized into nine Health Regions as the geographic space constituted by neighboring municipalities. Each Health Region has a major city. The population and their coordinates were obtained from IBGE. They are described in Table 2.

Since there are missing records for some municipalities, the daily records were temporal aggregated using a weekly average. This approach reduces the

Table 2 Municipality and coordinates used as representative of each Health Region.

Heath Region	Municipality	latitude	longitude
Baia da Ilha Grande	Angra dos Reis	-23.0011	-44.3196
Baixada Litorânea	Cabo Frio	-22.8894	-42.0286
Centro-Sul	Três Rios	-22.1165	-43.2185
Médio Paraíba	Volta Redonda	-22.5202	-44.0996
Metropolitana I	Rio de Janeiro	-22.9129	-43.2003
Metropolitana II	São Gonçalo	-22.8268	-43.0634
Noroeste	Itaperuna	-21.1997	-41.8799
Norte	Campos dos Goytacazes	-21.7622	-41.3181
Serrana	Petrópolis	-22.5200	-43.1926

everyday variation, negative values (decreasing number of notifications), and delays in notifications (weekend deaths notified only on Mondays). We use data from the first year of the pandemic (i.e., the first 52 weeks of the pandemic), starting on 5 March 2020.

We apply the percentage variation to smooth the data, with a data discretization into three ranges (a, b, and c), corresponding to decrease, stability, and increase in the number of deaths, respectively.

4.2.1 Experimental Setup

The experiments were performed using different combination of values for the input parameters (σ, β, and γ). The σ values varied with step of 0.5 in the range $[1, 3.5]$. For β, the integer values in the range $[2, 8]$ were used, and for γ, the values 0.6, 0.8 and 1.0.

Trend reversal moments in the death curve can be useful for monitoring disease behavior. In this way, sequences of events with distinct elements (i.e. high entropy) are more important than sequences of equal elements. So, after running tests using different combinations of values for the input parameters, the configuration with $\gamma = 0.6$, $\beta = 2$, and $\sigma = 2$ was selected because it is the one with highest amount of sequences with maximum entropy.

4.2.2 Results

Table 3 presents the sequence size, the number of SRGs, the number of SRG with max entropy, and the number of SRG with steady oscillation (two or more "a" or "c") for the results obtained from the chosen configuration ($\gamma = 0.6$, $\beta = 2$ and $\sigma = 2$).

The results of size three were ordered by entropy and number of occurrences. From the first four we picked two SRG that represent specific moments of the pandemic. The first one is a sequence "bcc", which expresses a significant rise. It starts at epidemiological week 3, from March the 27th to April 2nd of 2020, ending at epidemiological week 10. The same sequence starts at different weeks in different Health Regions: "Metropolitana I" and "Serrana" at

Table 3 Results for the best parameters configuration. Elements in column "steady oscillation" can include elements in column "SRGs with max entropy".

sequence size	**number of SRGs**	**SRGs with max entropy**	**steady oscillation (two or more "a" or "c")**
3	48	8	36
4	59	17	47
5	62	16	49
6	64	18	51
7	62	24	54
8	64	36	58
9	24	3	19
10	19	3	15
11	11	2	10
12	5	1	5
13	3	1	3
14	1	1	1
TOTALS:	**443**	**130**	**348**

week 3 propagating to "Baixada Litorânea" and "Noroeste" at week 4 reaching "Centro-Sul" at week 5. Then, again in "Serrana" at week 7, "Noroeste" at week 9, "Metropolitana I" and "Centro-Sul" at week 10.

Such results show the disease's interiorization and that disease peaks spread between health regions within a few weeks of difference. The municipality of Rio de Janeiro, as the state capital, has the largest population, an international airport, and a port. It is a major tourist center, attracting tourists from all over the world. It is not surprising that interiorization begins with the health region that contains it, spreading to other health regions with less population density.

The second SRG we picked to show is formed by the "abc" sequence, expressing a change of behavior, starting with a fall, changing to stability, and then rising. It starts at epidemiological week 37, from November the 22nd to 28th of 2020, ending at epidemiological week 42, from December the 28th of 2020 to January the 2nd. It starts at "Norte" Health Region at week 37, propagates to "Noroeste" at week 38, to "Metropolitana I" and "Centro-Sul" at week 41, and finally gets to "Baixada Litorânea" at week 42. Based on the population's preference for short trips and visits to the nearest tourist cities during the pandemic, it is possible to observe the spread of the disease in these places on national holidays at the end of 2020, and, consequently, the evolution of the disease in metropolitan regions and/ or close to them, after returning from travel.

Figure 7 shows the map of Rio de Janeiro state divided into Health Regions and the STSs (mean deaths per week) of each one in the same color as the map. The occurrence of both SRGs are marked as vertical lines, the first one in blue, and the second one in red at the health regions where they occur.

The results obtained from the COVID-19 spread and seismic domain show the versatility of the technique used, being able to find spatial-temporal patterns, with more than one dimension in space, in different domains.

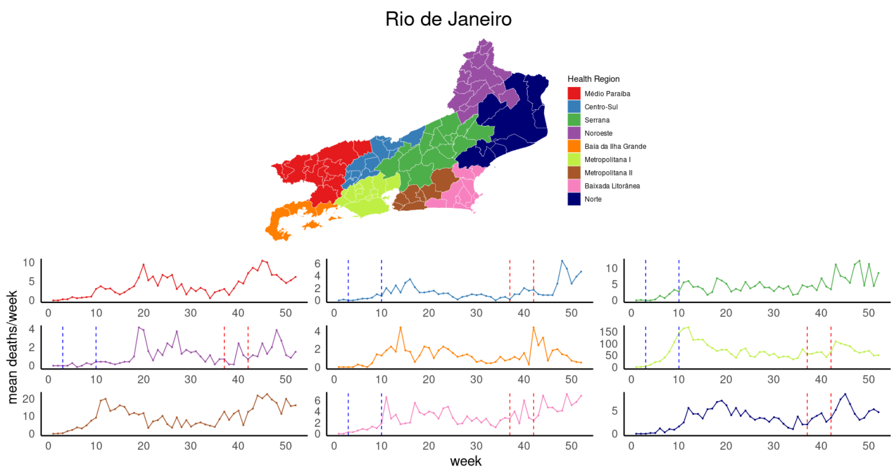

Fig. 7 State of Rio de Janeiro divided in Health Regions and their respective STS of mean deaths per week.

5 Conclusion

This paper introduced algorithm G-STSM for the problem of discovering tight space-time sequences. To the best of our knowledge, this is the first algorithm to find sequences constrained in space and time, without previous restrictions, that work with one dimension of time and three dimensions of space.

In its first experiment, using a real-world seismic dataset, G-STSM was compared using classification metrics with the SPADE + DBSCAN approach. It presented better performance being as much as three times faster. Furthermore, its behavior and outputs have been detailed with different parameter configurations and dataset sizes. In its second experiment, using a real-world health dataset, G-STSM was able to capture Covid-19 propagation patterns.

With these two experiments using different domains, we could show that G-STSM can extract constrained spatial-time sequences from different spatiotemporal datasets. Thus, G-STSM proved to be an efficient data mining tool for finding tight space-time sequences.

In future work, it would be positive to check when sequences can be predictors of new sequences.

Acknowledges

The authors would like to thank CAPES (finance code 001), CNPq, and FAPERJ for partially funding this paper.

References

Alatrista-Salas, H., Azé, J., Bringay, S., Cernesson, F., Selmaoui-Folcher, N., and Teisseire, M. (2015). A knowledge discovery process for spatiotemporal data: Application to river water quality monitoring. *Ecological Informatics*, 26(P2):127–139.

Alatrista-Salas, H., Bringay, S., Flouvat, F., Selmaoui-Folcher, N., and Teisseire, M. (2016). Spatio-sequential patterns mining: Beyond the boundaries. *Intelligent Data Analysis*, 20(2):293–316.

Aydin, B. and Angryk, R. (2016). Spatiotemporal event sequence mining from evolving regions. In *Proceedings - International Conference on Pattern Recognition*, volume 0, pages 4172–4177.

Aydin, B., Boubrahimi, S., Kucuk, A., Nezamdoust, B., and Angryk, R. (2020). Spatiotemporal event sequence discovery without thresholds. *GeoInformatica*.

Batu, B., Temizel, T., and Duzgun, H. (2017). A Non-Parametric Algorithm for Discovering Triggering Patterns of Spatio-Temporal Event Types. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2629–2642.

Ben Chaabene, S., Yeferny, T., and Ben Yahia, S. (2021). A roadside unit deployment framework for enhancing transportation services in maghrebian cities. *Concurrency Computation*, 33(1). cited By 2.

Brasil.IO (2021). Brasil.IO. Technical report, https://brasil.io.

Buchta, C., Hahsler, M., and Diaz, w. c. f. D. (2020). arulesSequences: Mining Frequent Sequences. Technical report, https://cran.r-project.org/web/packages/arulesSequences/index.html.

Campisano, R., Borges, H., Porto, F., Perosi, F., Pacitti, E., Masseglia, F., and Ogasawara, E. (2018). Discovering tight space-time sequences. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11031 LNCS, pages 247–257.

Chen, B.-H., Chuang, A.-W., and Chuang, K.-T. (2015). Discovery of Spatiotemporal chaining patterns. In *ACM International Conference Proceeding Series*, volume 07-09-Ocobert-2015.

Chen, B.-H., Teng, S.-Y., and Chuang, K.-T. (2017). Mining spatio-temporal chaining patterns in non-identity event databases. *Intelligent Data Analysis*, 21(S1):S71–S102.

Chen, H., Yu, S., Huang, F., Zhu, B., Gao, L., and Qian, C. (2020). Spatiotemporal Analysis of Retail Customer Behavior based on Clustering and Sequential Pattern Mining. In *2020 3rd International Conference on Artificial Intelligence and Big Data, ICAIBD 2020*, pages 284–288.

Chen, X., Pang, J., and Xue, R. (2014). Constructing and comparing user mobility profiles. *ACM Transactions on the Web*, 8(4).

Cheng, L., Yang, X., Tang, L., Duan, Q., Kan, Z., Zhang, X., and Ye, X. (2020). Spatiotemporal analysis of taxi-driver shifts using big trace data. *ISPRS International Journal of Geo-Information*, 9(4).

Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231, Portland, Oregon. AAAI Press.

Feuerhake, U. and Sester, M. (2013). Mining group movement patterns. In *GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*, pages 510–513.

Flamand, C., Fabregue, M., Bringay, S., Ardillon, V., Quénel, P., Desenclos, J., and Teisseire, M. (2014). Mining local climate data to assess spatiotemporal dengue fever epidemic patterns in French Guiana. *Journal of the American Medical Informatics Association : JAMIA*, 21(e2):e232–240.

Gurram, S. and Rama Mohan Reddy, A. (2014). A grid-based algorithm for mining spatio-temporal sequential patterns. *International Review on Computers and Software*, 9(4):659–666.

Hahsler, M., Piekenbrock, M., and Doran, D. (2019). Dbscan: Fast density-based clustering with R. *Journal of Statistical Software*, 91.

He, Z., Deng, M., Cai, J., Xie, Z., Guan, Q., and Yang, C. (2020). Mining spatiotemporal association patterns from complex geographic phenomena. *International Journal of Geographical Information Science*, 34(6):1162–1187.

Huang, Q., Li, Z., Li, J., and Chang, C. (2016). Mining frequent trajectory patterns from online footprints. In *Proceedings of the 7th ACM SIGSPATIAL International Workshop on GeoStreaming, IWGS 2016*.

Ibrahim, R. and Shafiq, M. (2019). Detecting taxi movements using Random Swap clustering and sequential pattern mining. *Journal of Big Data*, 6(1).

Julea, A., Méger, N., Bolon, P., Rigotti, C., Doin, M.-P., Lasserre, C., Trouvé, E., and Lăzărescu, V. (2011). Unsupervised spatiotemporal mining of satellite image time series using grouped frequent sequential patterns. *IEEE Transactions on Geoscience and Remote Sensing*, 49(4):1417–1430.

Julea, A., Méger, N., Trouvé, E., and Bolon, P. (2008). On extracting evolutions from satellite image time series. In *International Geoscience and Remote Sensing Symposium (IGARSS)*, volume 5, pages V228–V231.

Koseoglu, B., Kaya, E., Balcisoy, S., and Bozkaya, B. (2020). ST Sequence Miner: visualization and mining of spatio-temporal event sequences. *Visual Computer*, 36(10-12):2369–2381.

Lab, D. A. (2020). Generalized Discovery of Tight Space-Time Sequences \ Data Analytics Lab. Technical report, https://eic.cefet-rj.br/dal/generalized-discovery-of-tight-space-time-sequences/.

Lee, S., Lim, J., Park, J., and Kim, K. (2016). Next place prediction based on spatiotemporal pattern mining of mobile device logs. *Sensors (Switzerland)*, 16(2).

Leong, K. and Chan, S. (2012). STEM: A Novel approach for spatiotemporal sequence mining. *Asian Journal of Information Technology*, 11(3):94–99.

Li, K. and Fu, Y. (2014). Prediction of human activity by discovering temporal sequence patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(8):1644–1657.

Mooney, C. and Roddick, J. (2013). Sequential pattern mining - Approaches and algorithms. *ACM Computing Surveys*, 45(2).

OpendTect (2020). dGB Earth Sciences - Innovative Seismic Interpretations Solutions. Technical report, https://dgbes.com/.

Saleh, B. and Masseglia, F. (2008). Time aware mining of itemsets. In *2008 15th International Symposium on Temporal Representation and Reasoning*, pages 93–97. IEEE.

Shumway, R. H. and Stoffer, D. S. (2017). *Time Series Analysis and Its Applications: With R Examples*. Springer.

Sukanya, N. and Ranjit Jeba Thangaiah, P. (2019). Review on frequent patterns in data mining applications. *Journal of Advanced Research in Dynamical and Control Systems*, 11(4 Special Issue):1725–1730.

Sunitha, G. and Rama Mohan Reddy, A. (2014). Mining frequent patterns from spatiotemporal data sets: A survey. *Journal of Theoretical and Applied Information Technology*, 68(2):265–274.

Sunitha, G. and Rama Mohan Reddy, A. (2016). WRSP-miner algorithm for mining weighted sequential patterns from spatio-temporal databases. *Advances in Intelligent Systems and Computing*, 379:309–317.

Team, R. C. (2020). R: A Language and Environment for Statistical Computing. Technical report, https://www.R-project.org/, Vienna, Austria.

Tsoukatos, I. and Gunopulos, D. (2001). Efficient mining of spatiotemporal patterns. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 2121, pages 425–442.

Xu, L. and Kwan, M.-P. (2020). Mining sequential activity–travel patterns for individual-level human activity prediction using Bayesian networks. *Transactions in GIS*, 24(5):1341–1358.

Xue, C., Liu, J., Li, X., and Dong, Q. (2016). Normalized-mutual-information-based mining method for cascading patterns. *ISPRS International Journal of Geo-Information*, 5(10).

Yang, C. and Gidófalvi, G. (2018). Mining and visual exploration of closed contiguous sequential patterns in trajectories. *International Journal of Geographical Information Science*, 32(7):1282–1304.

Yusof, N. and Zurita-Milla, R. (2017). Mapping frequent spatio-temporal wind profile patterns using multi-dimensional sequential pattern mining. *International Journal of Digital Earth*, 10(3):238–256.

Zaki, M. (2001). SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1-2):31–60.

Zaki, M. J. (2000). Sequence mining in categorical domains: incorporating constraints. In *Proceedings of the ninth international conference on Information and knowledge management*, CIKM '00, pages 422–429, New York, NY, USA. Association for Computing Machinery.

Zhang, Z.-H., Zhai, W.-J., Shen, R.-P., Zeng, S.-C., and Min, F. (2018). State transition pattern with periodic wildcard gaps. In *Proceedings - 9th IEEE International Conference on Big Knowledge, ICBK 2018*, pages 440–447.