Open access • Journal Article • DOI:10.1109/TKDE.2007.190712

A Framework for Mining Sequential Patterns from Spatio-Temporal Event Data Sets

Yan Huang, Liqin Zhang, Pusheng Zhang

Institutions: Microsoft

Published on: 01 Apr 2008 - IEEE Transactions on Knowledge and Data Engineering (IEEE Computer Society)

Topics: Spatial database, Temporal database, Algorithmics, Spatial analysis and Transaction data

A Framework for Mining Sequential Patterns from Spatio-Temporal Event Data Sets

Yan Huang, *Member*, *IEEE*, Liqin Zhang, and Pusheng Zhang, *Member*, *IEEE*

Abstract—Given a large spatio-temporal database of events, where each event consists of the fields *event ID*, *time*, *location*, and *event type*, mining spatio-temporal sequential patterns identifies *significant* event-type sequences. Such spatio-temporal sequential patterns are crucial to the investigation of spatial and temporal evolutions of phenomena in many application domains. Recent research literature has explored the sequential patterns on transaction data and trajectory analysis on moving objects. However, these methods cannot be directly applied to mining sequential patterns from a large number of spatio-temporal events. Two major research challenges still remain: 1) the definition of significance measures for spatio-temporal sequential patterns to avoid spurious ones and 2) the algorithmic design under the significance measures, which may not guarantee the downward closure property. In this paper, we propose a sequence index as the significance measure for spatio-temporal sequential patterns, which is meaningful due to its interpretability using spatial statistics. We propose a novel algorithm called Slicing-STS-Miner to tackle the algorithmic design challenge using the spatial sequence index, which does not preserve the downward closure property. We compare the proposed algorithm with a simple algorithm called STS-Miner that utilizes the weak monotone property of the sequence index. Performance evaluations using both synthetic and real-world data sets show that the Slicing-STS-Miner is an order of magnitude faster than STS-Miner for large data sets.

Index Terms—Spatio-temporal sequential pattern, density ratio, sequence index, slicing, performance.

1 Introduction

Space and time are pervasive in our lives, and technology is changing the way that they are tracked. With the advances of technologies such as GPS, remote sensing, RFID, indoor locating devices, and geosensor networks, we can track spatio-temporal phenomena with increasingly finer spatial resolutions for longer periods of time. Scholars have been developing models to capture relationships among events in space and time for a long time [1]. Now, we cannot only observe relationships among specific events but also generalize spatio-temporal patterns from global or regional observations by using all events.

An event is described as a spatial and temporal happening. It happens at a given place and time and belongs to a specific event type. An event type categorizes or groups events that have the same characteristics together. Example event types include car accident, traffic jam, chromium-6-polluted water source, the West Nile disease, and deforestation. Events are the actual happenings of these types. For example, a car accident last Friday at the intersection of highways 75 and I35W is an event, whereas a car accident is an event type. Many events interact with each other and tend to exhibit spatial and temporal patterns, which may be summarized in the event-type level. It is critical to characterize such interactions involving space and time together among event types and identify them from large spatiotemporal data sets efficiently in various application domains. These domains include earth science, epidemiology, ecology, and climatology [1], [2], [3].

In this paper, we investigate the problem of finding spatio-temporal sequential patterns from a data set consisting of a large number of spatio-temporal events [4]. A spatio-temporal sequential pattern is a sequence of **event types**, with one leading to another. A sequential pattern of spatial and temporal event types in the form of $f_1 \rightarrow f_2 \rightarrow \ldots \rightarrow f_k$ represents a "chain reaction" from event type f_1 to event type f_2 and then to event type f_3 until it reaches event type f_k. This "chain reaction" happens in a specific spatial and temporal manner. For example, in epidemiology, some disease transmission may follow paths connecting different type of species. For example, the West Nile transmits from one species to another. Birds with the West Nile virus transmit the disease to nearby mosquitoes when they feed on infected birds and, then, the virus may be injected by mosquitoes into human beings in a nearby region, where it can multiply and possibly cause illness [5]. When many occurrences of such disease transmission from bird to mosquitoes and then to human beings are observed, one may conclude the West Nile sequential transmission path as "$\mathrm{Bird} \rightarrow \mathrm{Mosquito} \rightarrow \mathrm{Human\ Being}$." We distinguish an event type from their events. For example, the "Bird" in the sequential pattern represents an event-type bird and not an occurrence/event of a particular bird affected by the West Nile disease.

1.1 Problem Statement

Formally, the problem of mining spatio-temporal sequential patterns from spatio-temporal events is defined as follows: Let $\mathcal{F} = \{f_1, f_2, \ldots, f_K\}$ be a set of event types and $\mathcal{D}$ be a

- *Y. Huang and L. Zhang are with the Department of Computer Science and Engineering, University of North Texas, PO Box 311366, Denton, TX 76203. E-mail: {huangyan, lz0007}@unt.edu.*
- *P. Zhang is with Microsoft Corporation, One Microsoft Way, Redmond, WA 98028. E-mail: pzhang@microsoft.com.*

Manuscript received 8 Oct. 2006; revised 20 June 2007; accepted 8 Oct. 2007; published online 17 Oct. 2007.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0467-1006.
Digital Object Identifier no. 10.1109/TKDE.2007.190712.

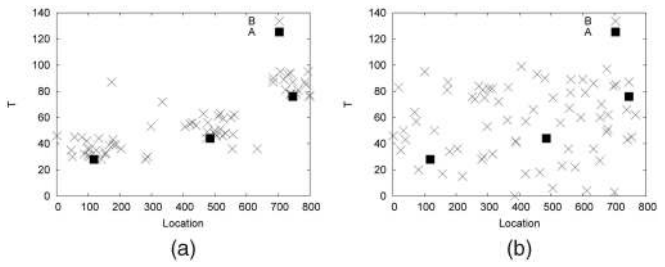


Fig. 1. A and B are two event types. (a) B follows A. (b) B is independent of A.

database of spatio-temporal events. Each event in $\mathcal{D}$ consists of the fields *event ID*, *time*, *location*, and *event type*, where *event type* $\in \mathcal{F}$. We denote the set of events that are associated with an event type $f \in \mathcal{F}$ as $f.\mathcal{E}$. The problem of mining spatio-temporal sequential patterns from $\mathcal{D}$ is finding all the *significant* event-type sequences in the form of $f_1 \rightarrow f_2 \ldots \rightarrow f_k$. In the West Nile example, the event-type set $\mathcal{F} = \{\ldots, Bird\ West\ Nile, Mosqutoes\ West\ Nile, Human\ Beings\ West\ Nile, \ldots\}$, where "..." represents other event types in the spatio-temporal databases, for example, other species affected by the West Nile. The event set $Bird\ West\ Nile.\mathcal{E}$ of event type $Bird\ West\ Nile$ includes all of the occurrences of the bird West Nile infection with their location and time information.

The first challenge in mining spatio-temporal sequential patterns is identifying a statistical *significance* measure to distinguish meaningful patterns from spurious ones. In Fig. 1, two types of events A and B are represented by symbols ■ and $\times$, respectively. The x-axis represents the one-dimensional space, and the y-axis represents the time. A visual inspection may reveal that events of B tend to occur around and after events of A in Fig. 1a, and events of B are independently distributed from events of A in Fig. 1b: B is denser in A's spatiotemporal neighborhood in Fig. 1a, and B is about equally distributed in A's neighborhood and elsewhere. Quantifying such observation means defining meaningful spatiotemporal statistical significance measures for sequential patterns such as $A \rightarrow B$ and longer sequential patterns such as $A \rightarrow B \rightarrow C$.

The second challenge is designing efficient algorithms to identify significant spatio-temporal sequential patterns from a massive number of spatio-temporal events. A spatio-temporal event type may appear multiple times in a sequential pattern. For example, $A \rightarrow B \rightarrow A$ is a valid sequence. An example of such a pattern may be that poverty leads to deforestation, which is due to the lack of resources in some regions of the world, which leads to increased poverty in return. For K event types and N events, the length of a potential sequential pattern is not bounded by K due to the repetitions of event types in the sequential pattern, for example, $A \rightarrow A \rightarrow \ldots \rightarrow A$. However, no sequential pattern is longer than N, since we only have N events. A naive approach to scan the database once for each pattern and check its significance is computationally prohibitive. A downward closure property is frequently used in data mining algorithms [6], [7], [8], [9] to reduce searching space. However, many meaningful significance measures in spatial and temporal application domains do not have the downward closure property in nature.

1.2 Related Work

The closely related work on spatio-temporal sequential pattern mining can be organized into two categories. The first is the sequential pattern mining [10] in the market basket data analysis and its generalization to pattern mining [7], [8] in time series data analysis. The second is the mining of trajectory patterns from historical spatio-temporal data.

Since introduced by Agrawal and Srikant [10], mining sequential patterns from market basket data has attracted much attention. It addressed the following problem: Given customers' purchasing records, find common purchased subsequences shared by a significant number of customers, for example, $(\mathrm{PC}, \mathrm{Internet\ service}) \rightarrow \mathrm{DVD} \rightarrow \mathrm{MP3\ player}$. Many efficient algorithms [6], [7], [8], [9] have been proposed to identify sequential patterns from market basket data. The sequential pattern mining from market basket data has been thus extended to explore sequential relationships from other sequence data, for example, medical treatments, Web log click streams, DNA sequences, and gene structures. In these applications, the notion of transaction or its substitute, which represents a collection of item sets, is essential. However, the "transactionization" of a spatiotemporal space to adopt the existing sequential pattern mining methods on market basket data may be unnatural due to the continuity of space and time [11]. The main issue with transactionization in spatial and temporal domains is that the spatial, temporal, or spatio-temporal relationships across partition/transaction boundaries may be lost in a disjoint partitioning, and relationships may be overcounted for an overlapping partitioning. Other sequential patterns in sequence data such as periodicity have also been explored in the context of transactions and time series databases [12], [13]. The repetitive events were identified in the data; however, the periodic pattern mining focused on the temporal aspects of the events with a fixed or moving window semantics and did not capture the sequential patterns involving space and time together.

Mamoulis et al. discussed mining, indexing, and querying of historical spatiotemporal data in [14], [15], and [16]. The trajectory of a moving object is typically a collection of consecutive spatial signatures at different time stamps. Retrieving similar trajectories might reveal underlying traveling patterns of moving objects in the data. Example applications include animal mobility investigation, storm tracing, and mobile phone tracing. In their context, trajectories of objects are given for investigations. Models and algorithms for mining frequent periodic subtrajectories were proposed. Each subtrajectory consists of a sequence of frequently visited places on trajectories. This work is related but different from ours. Trajectory is a collection of stops of the *same* moving object at different spatial locations. Trajectory analysis can be applied only if the trajectories have been provided a priori. In our context of spatio-temporal sequential pattern mining, the spatio-temporal data is a collection of different events, with each event belonging to one particular event type, as shown in Fig. 1. Thus, the notion of trajectory does not exist for spatio-temporal sequential patterns in this context. To the best of our knowledge, although there is a relatively large literature on spatial, temporal, and spatio-temporal data mining, little related work exists on mining sequential patterns from large spatio-temporal event data sets without trajectories given as a prior.

1.3 Our Contributions

In this paper, we focus on addressing the two aforementioned challenges of mining sequential patterns from large spatio-temporal data sets. The paper brings together the following contributions:

- We propose a framework for mining sequential patterns from spatio-temporal event data sets. The central piece of the framework is to use a *sequence index* as the significance measure for spatio-temporal sequential patterns. We define a *density ratio* for size two sequences and then extend it to *sequence index* for longer sequences. In addition, we establish the statistical interpretation for the proposed sequence index.
- We propose a novel algorithm called *Slicing-STS-Miner* for sequential pattern mining on spatio-temporal event data sets. The proposed algorithm tackles the algorithmic challenges by using the sequence index, which does not guarantee the downward closure property. *Slicing-STS-Miner* uses temporal slicing to partition the data set into overlapping slices according to time when the number of events is too large to be processed in memory. It processes each slice separately and utilizes the unidirectional property of time to recover the patterns across slice boundaries with low cost. *Slicing-STS-Miner* is complete and correct; that is, there are no false droppings or false admissions of sequential patterns.
- We analyze the algebraic costs of the algorithms and also experimentally evaluate the performance of the proposed algorithm against a simple algorithm called *STS-Miner*, which utilizes the weak monotone property of the sequence index on both synthetic and real-world data sets. The experimental results show that the *Slicing-STS-Miner* is scalable and is an order of magnitude faster than *STS-Miner* for large data sets.

This paper subsumes the conference version of the paper by providing detailed algebraic cost models for the algorithms and more extended experimental results. Many examples for illustrating the key definitions and algorithms are provided as well. The rest of this paper is organized as follows: Section 2 introduces the sequence index as the significance measure for spatio-temporal sequential patterns. The *Slicing-STS-Miner* and the *STS-Miner* algorithms for spatio-temporal sequential pattern mining are proposed in Section 3. Experimental evaluations using synthetic and real-world data sets are presented in Section 4. We summarize our work and discuss future directions in Section 5.

2 Significance Measure

In order to identify sequential patterns, the first issue that we need to address is how we can define the significance of a given spatio-temporal sequence of event types. For simplicity of illustration, we assume that the dimensionalities of both space and time are one in this paper. The definitions and algorithms can be easily generalized into higher dimensions for space and time as well in the targeted spatiotemporal application domains. In this paper, we use a spatio-temporal predicate called *follow* and its corresponding neighborhood definition to explain the framework and algorithms. However, our framework is general and can accommodate any other spatio-temporal predicate, as long as the predicate can be described by a well-defined neighborhood notion.

Informally, the *follow* spatio-temporal predicate describes that an event e_1 happens in the nearby region of another event e_2 shortly after e_2 happened. As shown in the example, the *follow* predicate requires a neighborhood definition to be precise.

Definition 1 (Follow). *We say that an event e' $follows_N$ a different event e, denoted by $e \rightarrow_N e'$, if and only if e' is in the spatio-temporal neighborhood $N(e)$ of e.*

A simple spatio-temporal neighborhood of an event e to capture the *follow* predicate can be defined as

$$N(e) = \{p | p \in \mathcal{D} \wedge distance(e.location, p.location) \leq \mathcal{R} \wedge (p.time - e.time) \in [0, \mathcal{T}]\}.$$

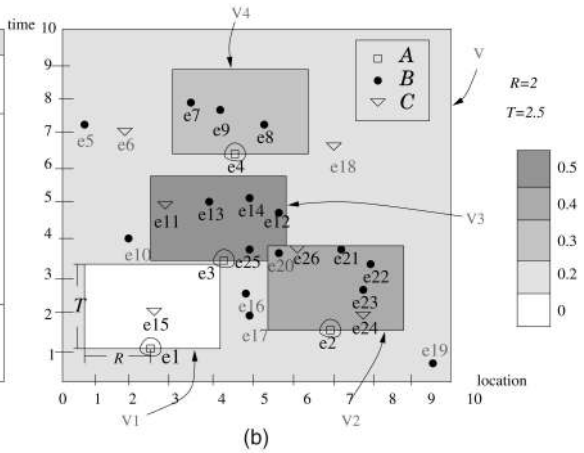
An example of the simple neighborhood could be a distance not more than 1.5 miles for any time interval of less than 1 hour; that is, $\mathcal{R} = 1.5$ and $\mathcal{T} = 1$. More complex spatio-temporal neighborhoods will be discussed later in Section 2.4. When the neighborhood definition for the *follow* predicate is clear from the context and causes no confusion, we abbreviate $e \rightarrow_N e'$ to $e \rightarrow e'$.

In Fig. 2, three spatio-temporal event types are represented using the symbols square (A), circle (B), and triangle (C), respectively. Their events are embedded in a spatio-temporal space. The x-axis denotes the linear space, and the y-axis denotes the linear time. The event set for each event type is listed in a table (Fig. 2a). Among many others, event e_{11} follows event e_3; that is, $e_3 \rightarrow e_{11}$ for the given distance $\mathcal{R}$ and time interval $\mathcal{T}$, as shown in the figure. Note that spatial distance $\mathcal{R}$ can be applied to both left and right directions in space, whereas the temporal interval $\mathcal{T}$ can be only applied to an upward direction due to the unidirectional property of time (and this property will be utilized when we design efficient algorithms using temporal slicing in Section 3).

One way of defining the significance measure of a pattern $f_1 \rightarrow f_2$ is to calculate the number of events of type f_2 that follow events of type f_1. However, this approach ignores the distribution of events of f_2 in the overall space. For example, in Fig. 1b, when event type B has many events, this naive significance measure will mistakenly conclude that since there are many events of B following events of A, then $A \rightarrow B$. The main problem of simple counting-based measures is that they do not compare with independent distribution and provide little guidance on how a proper threshold for the counts can be determined to define significance. As a result, spurious sequences may be detected as significant sequential patterns, especially when the number of events of involved event types is large. Instead of using a simple counting mechanism, we propose a sequence index as a significance measure for spatio-temporal sequential patterns, which may be more meaningful due to its interpretability by using

Event Type	Symbol	Event Set (Location, Time)
A	□	e1(2.55, 1.05), e2(6.90, 1.55), e3(4.20, 3.60), e4(4.50, 6.40)
B	•	e5(0.70, 7.20), e7(3.50, 7.90), e8(5.20, 7.20), e9(4.20, 7.70), e10(2.00, 4.00), e12(5.60, 4.70), e13(3.90, 5.00), e14(4.90, 5.10), e16(4.80, 2.60), e17(4.90, 1.90), e19(9.55, 0.60), e20(5.60, 3.70), e21(7.20, 3.80), e22(7.90, 3.45), e23(7.80, 2.80), e25(4.90, 3.80)
C	▽	e6(1.95, 7.00), e11(2.90, 4.90), e15(2.60, 2.00), e18(7.00, 6.60), e24(7.80, 1.95), e26(6.10, 3.80)

(a)



(b)

Fig. 2. (a) A sample spatio-temporal data set. (b) Densities of events of B in events of A's neighborhoods and, overall, are represented by shades of different intensities.

spatial statistics [17]. We will first introduce a density ratio as the significance measure for two event sets. We will then introduce the sequence index to measure sequential patterns with an arbitrary number of event types.

2.1 Density Ratio

We define the concept of *density* as a precursor to the definition of *density ratio*. The density of an event set in a selected spatio-temporal space V is the average number of events in a unit of V, that is, the total number of events of the event set divided by the volume $|V|$ of the selected spatio-temporal space V. The selected spatio-temporal space may be the whole embedding space or some arbitrary neighborhood, for example, the neighborhood of an event. In Fig. 2, the density of the event set of the event type B in the whole spatio-temporal embedding space V is $\frac{16}{|V|} = 0.16$, since $|V| = 10 \times 10 = 100$, which means that in each unit of the spatio-temporal embedding space, we expect to find 0.16 events of type B. The density for the event set of the event type B in region V_4 is $\frac{3}{|V_4|} = 0.3$, where $|V_4|$ is the volume of the neighborhood of e_4. Given $\mathcal{R} = 2$ and $\mathcal{T} = 2.5$, $|V_4| = 2 \times 2 \times 2.5 = 10$. The formal definition of density is given as follows:

Definition 2 (Density). *For a given spatio-temporal space V, the density of an event set E in V is the average number of events from event set E located in each unit of V, that is, $density(E, V) = \frac{|\{e|e \in E \wedge e \ inside \ V\}|}{|V|}$, where $|V|$ represents the volume of the space V, and $|\{e|e \in E \wedge e \ inside \ V\}|$ denotes the cardinality of the set of events that are in both E and V.*

Using the density concept, the density of an event type f in space V may be represented by $density(f.\mathcal{E}, V)$, where $f.\mathcal{E}$ is the event set of f.

Given two event sets E and E', if the average density of E' in the neighborhoods of events in E is higher than the density of E' in the overall embedding space, then it is likely that events in E' tend to follow events in E. In Fig. 2, the density $density(B.\mathcal{E}, V)$ of events of type B in the overall embedding space V is 0.16, as calculated before. The densities of events of type B that *follow* the events of A are

$$density(B.\mathcal{E}, V_1) = \frac{0}{2 \times 2 \times 2.5} = 0,$$
$$density(B.\mathcal{E}, V_2) = \frac{4}{2 \times 2 \times 2.5} = 0.4,$$
$$density(B.\mathcal{E}, V_3) = \frac{5}{2 \times 2 \times 2.5} = 0.5, \text{ and}$$
$$density(B.\mathcal{E}, V_4) = \frac{3}{2 \times 2 \times 2.5} = 0.3.$$

The average density of events of type B that *follow* events of A is $\frac{0+0.4+0.5+0.3}{4} = 0.3$. This means that in each unit of the spatio-temporal neighborhoods of events of A, we expect to see 0.3 events of B. Compared with the overall density of events of B in the whole embedding space, the density ratio of these two densities is $\frac{0.3}{0.16}$ and is greater than 1. This means that it is likely that $A \rightarrow B$. To visually represent the notion of density, the densities for events of type B in the whole space V and in the neighborhoods of each event of type A, that is, $V_1, \ldots, V_4$, are computed and represented with different shades of gray in Fig. 2. Formally, the density ratio concept is defined as follows:

Definition 3 (Density Ratio for Two Event Sets). *For two event sets E and E' and a given $follow(\rightarrow)$ predicate defined by a neighborhood function $N(e)$ for any event $e \in E$, the density ratio of E and E' under the given follow predicate $\rightarrow$ is defined as*

$$densityRatio(E \rightarrow E') = \frac{average_{e \in E}(density(E', N(e)))}{density(E', V)},$$

where V is the spatio-temporal embedding space.

Note that the $follow$ $(\rightarrow)$ relationship is an overloaded predicate in this paper. An event e' may follow another event e as defined in Definition 1, and an event set E' may follow another event set E with some density ratio as defined in Definition 3. Furthermore, an event e may follow a sequence of events, which will be defined shortly. Similar to object-oriented programming, the meaning of the overloaded *follow* predicate is decided by the types of the variables associated with the predicate.

The numerator represents the average density of the event set E' that *follow* some event in event set E. The denominator of the ratio is the base density of event set E'

in the embedding space. Using the notation of density ratio for two event sets, the density ratio between the event sets of two event types f and f' is

$$densityRatio(f.\mathcal{E} \rightarrow f'.\mathcal{E}) = \frac{average_{e \in f.\mathcal{E}}(density(f'.\mathcal{E}, N(e)))}{density(f'.\mathcal{E}, V)}.$$

If the density of events of f' following events of f is higher than f''s overall density, the density ratio will be greater than 1, and event type f' *follows* event type f to some degree, indicated by how far the ratio is from 1. On the contrary, if the density of events of type f' following events of f is lower than f''s overall density, then the density ratio will be less than 1, and event type f repels event type f' to some degree, as expressed by the ratio. Otherwise, the density ratio is close to 1, and event types f and f' are independently distributed. Typically, for an unknown statistical distribution, Monte Carlo simulations are used to create a confidence envelope to distinguish independence from patterns. The confidence envelope gives guidance on how far the sequence index should be from 1 for a candidate to be considered as a pattern. Designing the Monte Carlo simulation for sequential patterns is an interesting future work and is a research topic in its own right but is beyond the scope of this paper.

2.2 Sequence Index

A spatio-temporal sequence of k event types is called a (k)-sequence. We refer to the subsequence consisting of the ith to the jth event types of a (k)-sequence $\overrightarrow{S}$ as $\overrightarrow{S}[i:j]$, with $i \in [1, k-1]$, $j \in [i+1, k]$, and the ith event type in the sequence $\overrightarrow{S}$ as $\overrightarrow{S}[i]$. Generalizing the density ratio concept to longer sequences $(k > 2)$ is nontrivial. For a sequence $\overrightarrow{S}$ to be significant, there are two requirements that a significance measure needs to meet:

1. The events of any event type $\overrightarrow{S}[i]$ in $\overrightarrow{S}$ need to *follow* an "event sequence" of the sequence $\overrightarrow{S}[1:i-1]$.
2. The density ratio between any two consecutive event types needs to be significant.

To meet the first requirement, we define the concept of event sequence and a tail event set of a (k)-sequence. A tail event set serves as a link to capture the *follow* relationship from a sequence to an event.

Definition 4 (Event Sequence). *An event sequence $\overrightarrow{E}$ of k-sequence $\overrightarrow{S}$ is a sequence of events such that 1) $\overrightarrow{E}[i]$ is of event type $\overrightarrow{S}[i]$ for $i \in [1, k]$ and 2) $\overrightarrow{E}[i] \rightarrow \overrightarrow{E}[i+1]$ for $i \in [1, k-1]$, where $\overrightarrow{E}[i]$ represents the ith event of $\overrightarrow{E}$.*

Definition 5 (An Event Follows an Event Sequence). *For an event sequence $\overrightarrow{E}$ of size k, another event e is said to follow the event sequence $\overrightarrow{E}$, represented by $\overrightarrow{E} \rightarrow e$, if $\overrightarrow{E}[k] \rightarrow e$.*

Definition 6 (Tail Event Set of a Sequence). *For a (k)-sequence $\overrightarrow{S}$, the set of events of type $\overrightarrow{S}[k]$ that participate in one of the event sequences of $\overrightarrow{S}$ is called the tail event set of $\overrightarrow{S}$ and is denoted as $tailEventSet(\overrightarrow{S})$.*

Lemma 1. *An event e is a tail event of a (k)-sequence $\overrightarrow{S}$ if and only if e is of event type $\overrightarrow{S}[k]$ and e follows a tail event of $\overrightarrow{S}[k-1]$.*

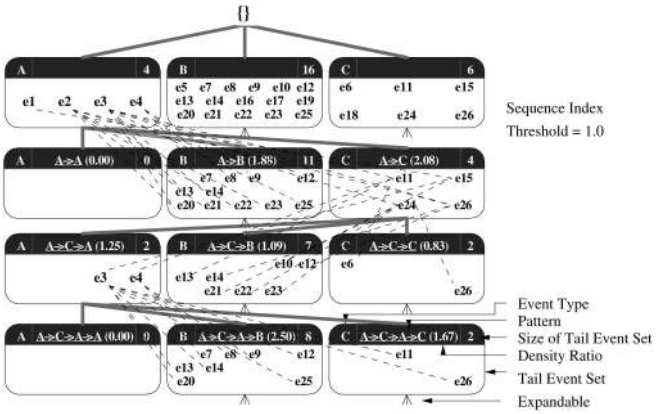


Fig. 3. Partial sequential pattern tree for the sample data in Fig. 2.

Proof. Let e be a tail event of $\overrightarrow{S}$. According to the definition of tail event, e is of event type $\overrightarrow{S}[k]$, and there is an event sequence $\overrightarrow{E}$ of $\overrightarrow{S}$ that e participates. Then, $\overrightarrow{E}[k-1]$ is a tail event of $\overrightarrow{S}[k-1]$ that e *follows*.

Now, let e_2 be of event type $\overrightarrow{S}[k]$ and *follows* a tail event e_3 of $\overrightarrow{S}[k-1]$. Let $\overrightarrow{E}_2$ be the event sequence of $\overrightarrow{S}[k-1]$ that e_3 participates in. Then, the event sequence by appending e_2 to $\overrightarrow{E}_2$ is an event sequence of $\overrightarrow{S}$. Thus, e_2 is a tail event of $\overrightarrow{S}$. □

Lemma 1 allows efficient algorithms that store tail event sets, instead of event sequences, to be developed.

Example 1. To illustrate the concepts, we created a sequence tree in Fig. 3, which represents a subset of the sequential patterns generated from the data set in Fig. 2. Each node in the tree represents a sequential pattern starting from the root. Each node may be expanded to a longer sequential pattern by branching to a new node labeled with event type A, B, or C. We have selectively expanded some of the nodes due to the lack of space. The events in each node are the tail events of the current pattern. The dotted lines between two events represent the *follow* relationship between the two linked events.

Considering $A \rightarrow C$ as a sequential pattern, the tail event set of $A \rightarrow C$ will be all of the events of C that follow some event of A; therefore,

$$tailEventSet(A \rightarrow C) = \{e_{11}, e_{15}, e_{24}, e_{26}\}.$$

Now, we append one more event type A to this pattern and consider $A \rightarrow C \rightarrow A$ as a sequential pattern. The tail event set of this pattern will be the events of A that follow the tail event set of $A \rightarrow C$. Among the four events of A, e_3 and e_4 follow some event in $\{e_{11}, e_{15}, e_{24}, e_{26}\}$. Thus, the tail event set of $A \rightarrow C \rightarrow A$ is $\{e_3, e_4\}$.

To meet the second requirement of the significance measure for a sequence $\overrightarrow{S}$, we define a *sequence index* as the minimal of the density ratios between the tail event set of an $(i-1)$-subsequence $\overrightarrow{S}[1:i-1]$ and the event set of event type $\overrightarrow{S}[i]$, that is,

$$densityRatio(tailEventSet(\overrightarrow{S}[1:i-1]) \rightarrow \overrightarrow{S}[i].\mathcal{E}))$$

for all $i \in [1, k-1]$. The basic rationale of choosing the minimal of the density ratios as the sequence index for a

sequential pattern such as $A \rightarrow B \rightarrow C$ is to ensure that the sequence is linked by a *follow* relationship with minimal "strength," as measured by the minimal of the density ratios. This prevents a sequence with very high density ratio for $A \rightarrow B$ but very low density ratio for $(A \rightarrow B) \rightarrow C$ to be selected as significant, as measured by the average of the density ratios. Furthermore, by choosing the minimal to be above θ, we implicitly require that the average is also above θ. Instead of using all the events in $\overrightarrow{S}[i-1].\mathcal{E}$, we use the tail event set of $\overrightarrow{S}[1:i-1]$ to compute the density ratio with $\overrightarrow{S}[i].\mathcal{E}$, because we want to measure the impact of the $(k-1)$-sequence on the event type $\overrightarrow{S}[i]$ instead of the impact of event type $\overrightarrow{S}[i-1]$. Now, we use density ratio to define sequence index.

Definition 7 (Sequence Index). *The sequence index of a* (k)-sequence $\overrightarrow{S}$ *is defined as follows:*

1. *When* $k = 2$,

$$seqIndex(\overrightarrow{S}) = densityRatio(\overrightarrow{S}[1].\mathcal{E} \rightarrow \overrightarrow{S}[2].\mathcal{E}).$$

2. *When* $k \geq 3$,

$$seqIndex(\overrightarrow{S}) = \min(seqIndex(\overrightarrow{S}[1:k-1],$$
$$densityRatio(tailEventSet(\overrightarrow{S}[1:k-1]) \rightarrow \overrightarrow{S}[k].\mathcal{E})).$$

Example 2. To compute the sequence index of $A \rightarrow C \rightarrow A$, we will use the tail event set of $A \rightarrow C$ to compute its density ratio with the event set $A.\mathcal{E}$ of event type A. As shown in the previous example, $tailEventSet(A \rightarrow C) = \{e_{11}, e_{15}, e_{24}, e_{26}\}$, with four events. Two events of A, namely, e_3 and e_4, follow $tailEventSet(A \rightarrow C)$. Note that in this example, e_3 of event type A follows e_{15}, and e_4 of event type A follows e_{11}. The average density of A around $tailEventSet(A \rightarrow C) = \{e_{11}, e_{15}, e_{24}, e_{26}\}$ is thus $\frac{1+1+0+0}{4\times2\times2\times2.5} = 0.05$ for a spatial distance of 2 and temporal interval of 2.5. Thus,

$$densityRatio(tailEventSet(A \rightarrow C) \rightarrow A.\mathcal{E}) = \frac{average_{e \in tailEventSet(A \rightarrow C)} density(A.\mathcal{E}, N(e))}{\frac{|A.\mathcal{E}|}{|V|}} = \frac{\frac{1+1+0+0}{4\times2\times2\times2.5}}{\frac{4}{100}} = 1.25.$$

Similarly, the density ratio $densityRatio(A \rightarrow C)$ between event types A and C can be calculated as 2.08. Therefore, $seqIndex = \min(1.25, 2.08) = 1.25$.

We say that a (k)-sequence $\overrightarrow{S}$ is a *significant* spatio-temporal sequence if and only if $seqIndex(\overrightarrow{S}) \geq \theta$, where θ is a user-given threshold (typically above 1). *The problem of mining spatio-temporal sequential patterns is finding all significant spatio-temporal sequences.*

2.3 Properties of the Density Ratio and Sequence Index

The density ratio for two event types f_1 and f_2 is close to 1 if f_1 and f_2 are independently distributed. A density ratio value significantly higher than 1 means positive influence (attracting), and a density value significantly less than 1 means negative influence (repelling).

For a *significant* k-sequence $\overrightarrow{S}$, although all of its subsequences in the form of $\overrightarrow{S}[1:i]$, where $i \in [2,k]$ are significant, there is no guarantee that other subsequences, for example, $\overrightarrow{S}[2:3]$, are significant. The following example illustrates this property.

Example 3. Considering the sequence $\overrightarrow{S} = A \rightarrow C \rightarrow A$, we show that its subsequence $C \rightarrow A$ has a smaller sequence index value and thus may not be significant, even if $\overrightarrow{S}$ is significant for some θ. In Example 2, we have shown that the $seqIndex(\overrightarrow{S}) = 1.25$. We know that among the six events of event type C, $e_{15} \rightarrow e_3$, and $e_{11} \rightarrow e_4$ (this information can be found in Table 1). Now,

$$seqIndex(C \rightarrow A) = densityRatio(C \rightarrow A) = \frac{average_{e \in C.\mathcal{E}} density(A.\mathcal{E}, N(e))}{\frac{|A|}{|V|}} = \frac{\frac{0+1+1+0+0+0}{6\times2\times2\times2.5}}{\frac{4}{100}} = 0.83 < 1.25.$$

Thus, for a sequence index threshold θ, where $0.83 < \theta < 1.25$, $\overrightarrow{S} = A \rightarrow C \rightarrow A$ is significant, but its subsequence $C \rightarrow A$ is not significant. Thus, sequence index is not antimonotone and does not guarantee the downward closure property for the sequential patterns.

Lemma 2 (Weak Antimonotone). *For a significant* k-sequence $\overrightarrow{S}$, *all of its subsequences in the form of* $\overrightarrow{S}[1:i]$, *where* $i \in [2,k]$ *are significant (the weak antimonotone property).*

Given the weak antimonotone property, although sequential pattern mining algorithms may still generate candidate $(k+1)$-sequences from significant k-sequences by appending one more event type at the end of a significant k-sequence, algorithms utilizing the downward closure property of the sequences, in general, could not be used. For example, if we generate $A \rightarrow B \rightarrow C$ only if $A \rightarrow B$ and $B \rightarrow C$ are significant, we may miss the significant sequence $A \rightarrow B \rightarrow C$ when $B \rightarrow C$ is not significant.

The density ratio may be interpreted using the relative density against the base density using spatial statistics and is similar to a spatial statistical measure called the cross-K function [17]. However, there are two important differences between the cross-K function and the sequence index. First, the sequence index can be applied to any sequence, whereas the cross-K function is only meaningful for (2)-sequences. Second, the sequence index measure is much more general than the cross-K function. It incorporates the time dimension by using the *follow* predicate. Other spatio-temporal predicates such as *during* may be defined using sequence index with suitable neighborhood functions for interval-based events to emulate more complex variance of spatio-temporal phenomena and their interactions.

TABLE 1
Algorithm Trace for the Sample Data in Fig. 2, with the Sequence Index Set to 1

Pattern	Join Result	Tail Event Set	Sequence Index
(A)→A			min(0.00)
(A)→B	$\{e_4\}\rightarrow e_7.\{e_4\}\rightarrow e_8.\{e_4\}\rightarrow e_9.\{e_3\}\rightarrow e_{12}.\{e_3\}\rightarrow e_{13}.\{e_3\}\rightarrow e_{14}.\{e_2, e_3\}\rightarrow e_{20}.\{e_2\}\rightarrow e_{21}.\{e_2\}\rightarrow e_{22}.\{e_2\}\rightarrow e_{23}.\{e_3\}\rightarrow e_{25}$	$e_7, e_8, e_9, e_{12}, e_{13}, e_{14}, e_{20}, e_{21}, e_{22}, e_{23}, e_{25}$	min(1.88)
(A→B)→A	$\{e_{12}, e_{13}, e_{14}\}\rightarrow e_4$	e_4	min(1.88,0.68)
(A→B)→B	$\{e_8, e_9\}\rightarrow e_7.\{e_{13}, e_{14}\}\rightarrow e_8.\{e_8\}\rightarrow e_9.\{e_{20}, e_{21}, e_{25}\}\rightarrow e_{12}.\{e_{12}, e_{20}, e_{25}\}\rightarrow e_{13}.\{e_{12}, e_{13}, e_{20}, e_{25}\}\rightarrow e_{14}.\{e_{20}, e_{22}, e_{23}\}\rightarrow e_{21}.\{e_{23}\}\rightarrow e_{22}.\{e_{20}\}\rightarrow e_{25}$	$e_7, e_8, e_9, e_{12}, e_{13}, e_{14}, e_{21}, e_{22}, e_{25}$	min(1.88,1.14)
(A→B)→C	$\{e_{13}\}\rightarrow e_6.\{e_{12}\}\rightarrow e_{18}.\{e_{20}, e_{22}, e_{23}\}\rightarrow e_{26}$	e_6, e_{18}, e_{26}	min(1.88,0.76)
(A)→C	$\{e_3\}\rightarrow e_{11}.\{e_1\}\rightarrow e_{15}.\{e_2\}\rightarrow e_{24}.\{e_2, e_3\}\rightarrow e_{26}$	$e_{11}, e_{15}, e_{24}, e_{26}$	min(2.08)
(A→C)→A	$\{e_{15}\}\rightarrow e_3.\{e_{11}\}\rightarrow e_4$	e_3, e_4	min(2.08,1.25)
(A→C→A)→A			min(2.08,1.25,0.00)
(A→C→A)→B	$\{e_4\}\rightarrow e_7.\{e_4\}\rightarrow e_8.\{e_4\}\rightarrow e_9.\{e_3\}\rightarrow e_{12}.\{e_3\}\rightarrow e_{13}.\{e_3\}\rightarrow e_{14}.\{e_3\}\rightarrow e_{20}.\{e_3\}\rightarrow e_{25}$	$e_7, e_8, e_9, e_{12}, e_{13}, e_{14}, e_{20}, e_{25}$	min(2.08,1.25,2.50)
(A→C→A→B)→A	$\{e_{12}, e_{13}, e_{14}\}\rightarrow e_4$	e_4	min(2.08,1.25,2.50,0.94)
(A→C→A→B)→B	$\{e_8, e_9\}\rightarrow e_7.\{e_{13}, e_{14}\}\rightarrow e_8.\{e_8\}\rightarrow e_9.\{e_{20}, e_{25}\}\rightarrow e_{12}.\{e_{12}, e_{20}, e_{25}\}\rightarrow e_{13}.\{e_{12}, e_{13}, e_{20}, e_{25}\}\rightarrow e_{14}.\{e_{20}\}\rightarrow e_{21}.\{e_{20}\}\rightarrow e_{25}$	$e_7, e_8, e_9, e_{12}, e_{13}, e_{14}, e_{21}, e_{25}$	min(2.08,1.25,2.50,1.25)
(A→C→A→B→B)→A	$\{e_{12}, e_{13}, e_{14}\}\rightarrow e_4$	e_4	min(2.08,1.25,2.50,1.25,0.94)
(A→C→A→B→B)→B	$\{e_8, e_9\}\rightarrow e_7.\{e_{13}, e_{14}\}\rightarrow e_8.\{e_8\}\rightarrow e_9.\{e_{21}, e_{25}\}\rightarrow e_{12}.\{e_{12}, e_{25}\}\rightarrow e_{13}.\{e_{12}, e_{13}, e_{25}\}\rightarrow e_{14}$	$e_7, e_8, e_9, e_{12}, e_{13}, e_{14}$	min(2.08,1.25,2.50,1.25,0.94)
(A→C→A→B→B)→C	$\{e_{13}\}\rightarrow e_6.\{e_{12}\}\rightarrow e_{18}$	e_6, e_{18}	min(2.08,1.25,2.50,1.25,0.42)
(A→C→A→B)→C	$\{e_{13}\}\rightarrow e_6.\{e_{12}\}\rightarrow e_{18}.\{e_{20}\}\rightarrow e_{26}$	e_6, e_{18}, e_{26}	min(2.08,1.25,2.50,0.63)
(A→C→A)→C	$\{e_3\}\rightarrow e_{11}.\{e_3\}\rightarrow e_{26}$	e_{11}, e_{26}	min(2.08,1.25,1.67)
(A→C→A→C)→A	$\{e_{11}\}\rightarrow e_4$	e_4	min(2.08,1.25,1.67,1.25)
(A→C→A→C→A)→A			min(2.08,1.25,1.67,1.25,0.00)
(A→C→A→C→A)→B	$\{e_4\}\rightarrow e_7.\{e_4\}\rightarrow e_8.\{e_4\}\rightarrow e_9$	e_7, e_8, e_9	min(2.08,1.25,1.67,1.25,1.88)
(A→C→A→C→A→B)→A			min(2.08,1.25,1.67,1.25,1.88,0.00)
(A→C→A→C→A→B)→B	$\{e_8, e_9\}\rightarrow e_7.\{e_8\}\rightarrow e_9$	e_7, e_9	min(2.08,1.25,1.67,1.25,1.88,0.63)
(A→C→A→C→A→B)→C			min(2.08,1.25,1.67,1.25,1.88,0.00)
(A→C→A→C→A)→C			min(2.08,1.25,1.67,1.25,0.00)
(A→C→A→C)→B	$\{e_{26}\}\rightarrow e_{12}.\{e_{11}\}\rightarrow e_{13}.\{e_{26}\}\rightarrow e_{14}$	e_{12}, e_{13}, e_{14}	min(2.08,1.25,1.67,0.94)
(A→C→A→C)→C	$\{e_{11}\}\rightarrow e_6$	e_6	min(2.08,1.25,1.67,0.83)
(A→C)→B	$\{e_{15}\}\rightarrow e_{10}.\{e_{26}\}\rightarrow e_{12}.\{e_{11}\}\rightarrow e_{13}.\{e_{26}\}\rightarrow e_{14}.\{e_{24}\}\rightarrow e_{21}.\{e_{24}\}\rightarrow e_{22}.\{e_{24}\}\rightarrow e_{23}$	$e_{10}, e_{12}, e_{13}, e_{14}, e_{21}, e_{22}, e_{23}$	min(2.08,1.09)
(A→C)→C	$\{e_{11}\}\rightarrow e_6.\{e_{24}\}\rightarrow e_{26}$	e_6, e_{26}	min(2.08,0.83)
(B)→A	$\{e_{16}, e_{17}\}\rightarrow e_3.\{e_{12}, e_{13}, e_{14}\}\rightarrow e_4$	e_3, e_4	min(0.78)
(B)→B	$\{e_8, e_9\}\rightarrow e_7.\{e_{13}, e_{14}\}\rightarrow e_8.\{e_8\}\rightarrow e_9.\{e_{16}, e_{20}, e_{21}, e_{25}\}\rightarrow e_{12}.\{e_{10}, e_{12}, e_{16}, e_{20}, e_{25}\}\rightarrow e_{13}.\{e_{12}, e_{13}, e_{20}, e_{25}\}\rightarrow e_{14}.\{e_{17}\}\rightarrow e_{16}.\{e_{16}, e_{17}\}\rightarrow e_{20}.\{e_{20}, e_{22}, e_{23}\}\rightarrow e_{21}.\{e_{23}\}\rightarrow e_{22}.\{e_{19}\}\rightarrow e_{23}.\{e_{16}, e_{17}, e_{20}\}\rightarrow e_{25}$	$e_7, e_8, e_9, e_{12}, e_{13}, e_{14}, e_{16}, e_{20}, e_{21}, e_{22}, e_{23}, e_{25}$	min(1.13)
(B)→C	$\{e_{13}\}\rightarrow e_6.\{e_{10}, e_{16}\}\rightarrow e_{11}.\{e_{12}\}\rightarrow e_{18}.\{e_{19}\}\rightarrow e_{24}.\{e_{16}, e_{17}, e_{20}, e_{22}, e_{23}\}\rightarrow e_{26}$	$e_6, e_{11}, e_{18}, e_{24}, e_{26}$	min(1.04)
(C)→A	$\{e_{15}\}\rightarrow e_3.\{e_{11}\}\rightarrow e_4$	e_3, e_4	min(0.83)
(C)→B	$\{e_6\}\rightarrow e_5.\{e_6\}\rightarrow e_7.\{e_{18}\}\rightarrow e_8.\{e_{15}\}\rightarrow e_{10}.\{e_{26}\}\rightarrow e_{12}.\{e_{11}\}\rightarrow e_{13}.\{e_{26}\}\rightarrow e_{14}.\{e_{24}\}\rightarrow e_{21}.\{e_{24}\}\rightarrow e_{22}.\{e_{24}\}\rightarrow e_{23}$	$e_5, e_7, e_8, e_{10}, e_{12}, e_{13}, e_{14}, e_{21}, e_{22}, e_{23}$	min(1.04)
(C)→C	$\{e_{11}\}\rightarrow e_6.\{e_{24}\}\rightarrow e_{26}$	e_6, e_{26}	min(0.56)

Patterns with sequence index underscored are terminal sequences.

2.4 Alternative Neighborhood Definition

Many applications require a dynamic neighborhood definition based on the time delay of two events. For example, in epidemiology, disease outbreak spreads spatially and diffuses when one moves away in time from the source event.

Thus, a more sensible way is to define the spatial neighborhood as a function of time delay. Let $\mathcal{R}$ be the maximal distance and $\mathcal{T}$ be the maximal time interval for two events to still be related. Then, for a given event e, its spatial neighborhood of e is a function f^N of time delay, that is, $N(e) = \{p|p \in \mathcal{D} \wedge distance(e.location,\ p.location) \leq f^N(p.time - e.time) \wedge \ \ (p.time - e.time) \in [0, \mathcal{T}]\}$. In the epidemiology example mentioned before, the required property of the spatio-temporal neighborhood function would be $\frac{\partial(f^N(p.time-e.time))}{\partial(p.time-e.time)} \leq 0$. This property states that with time passing by, e's spatial neighborhood stays constant or shrinks. For example, let e be a person identified with a contagious disease. Because of the mobility of the person at the early stage of the disease, many people in a large neighborhood may be affected. When time passes by, the mobility of the person decreases; thus, the potential impact region of the person decreases. When the derivative ratio is zero, $f^N(p.time - e.time) = \mathcal{R}$ is a constant function, and the distance remains $\mathcal{R}$ for all $(p.time - e.time) \in [0, \mathcal{T}]$. Fig. 4a shows the constant neighborhood function

$$N(e) = \{p|p \in \mathcal{D} \wedge distance(e.location, p.location) \leq \mathcal{R} \wedge (p.time - e.time) \in [0, \mathcal{T}]\},$$

Fig. 4b shows a linearly decreasing neighborhood function

$$N(e) = \{p|p \in \mathcal{D} \wedge distance(e.location, p.location) \leq \mathcal{R} \times \left(1 - \frac{t - e.time}{\mathcal{T}}\right) \wedge (p.time - e.time) \in [0, \mathcal{T}]\},$$

and Fig. 4c shows a neighborhood function

$$N(e) = \{p|p \in \mathcal{D} \wedge distance(e.location, p.location) \leq \mathcal{R} \times \sqrt{1 - \frac{t - e.time}{\mathcal{T}}} \wedge (p.time - e.time) \in [0, \mathcal{T}]\}$$

that decreases sublinearly.

Furthermore, the spatio-temporal neighborhood requirements for different event types may be different. Our framework can be extended to address this heterogeneity issue by adding nonuniform neighborhood definitions for different event types. Users will be asked to give neighborhood functions between different event types.

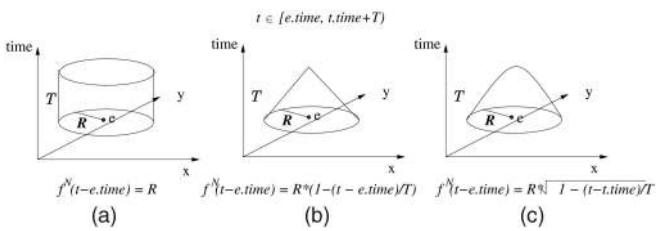


Fig. 4. Neighborhood functions. (a) Constant. (b) Linearly decreasing. (c) Sublinearly decreasing.

3 Mining Spatio-temporal Sequential Patterns

In this section, we propose a new algorithm called *Slicing-STS-Miner* for sequential pattern mining from large spatio-temporal event data sets. The proposed algorithm tackles the algorithmic design challenges by using the sequence index, which does not guarantee the downward closure property. We compare our proposed algorithm with an iterative pattern growth algorithm called *STS-Miner* that utilizes the weak antimonotone property of the sequence index. We first introduce the pattern growth algorithm *STS-Miner*.

3.1 STS-Miner

STS-Miner starts off with an empty sequence and generates all possible candidate $(k+1)$-sequences by attaching one more event type to a significant (k)-sequence. For simplicity of presentation, we assume the spatial neighborhood of a euclidean distance less than $\mathcal{R}$ and a temporal interval $\mathcal{T}$ for the sequential patterns. Neighborhoods can be parameterized and passed to the algorithms and procedures for more complex definitions. A sequential pattern tree that compresses all the sequential patterns is maintained. Each node $\overrightarrow{S}$ of depth k (the root is of depth 0) in the pattern tree represents a sequential pattern of size k starting from the root. The children of a node $\overrightarrow{S}$ are the sequential patterns that can be constructed by attaching one more event type to $\overrightarrow{S}$. The parent of $\overrightarrow{S}$ is $\overrightarrow{S}[1:k-1]$. In addition to the event type $\overrightarrow{S}[k]$, each node also maintains $\overrightarrow{S}$'s tail event set together with its size, and the density ratio dr and join size between the event set of event type $\overrightarrow{S}[k]$ and the tail event set of $\overrightarrow{S}$'s parent, that is, $\overrightarrow{S}[1:k-1]$. If dr is below a user-specified sequence index threshold, then $\overrightarrow{S}$ is a terminal node, which means that $\overrightarrow{S}$ will not be further expanded. Fig. 3 shows a sequential pattern tree partially expanded for up to four levels by using the data given in Fig. 2. The nodes that are not expanded and without a chicken foot attached are the terminal nodes for a sequential index threshold of 1.0. Among the three singleton sequential patterns A, B, and C, A's expansion to $A \rightarrow A$, $A \rightarrow B$, and $A \rightarrow C$ are shown. $A \rightarrow A$ is a terminal node, since its sequence index is 0. Then, $A \rightarrow C$'s expansion is shown, followed by that of $A \rightarrow C \rightarrow A$. The details of the running trace are shown in Table 1, where the full sequential pattern tree may be reconstructed.

To create such a sequential pattern tree, we will start with an initial tree with a null node linking to each event type, representing singleton sequential sequences. We call a recursive procedure $depthFirstExpand(p, \mathcal{D}, \theta)$ in a depth-first fashion for every singleton sequence p. For a node p and its tail event set $p.tailEventSet$, the procedure tries every event type in $\mathcal{D}$ to expand p. For each trial, it will compute the new density ratio and tail event set. The $join(E \rightarrow E')$ function in the $depthFirstExpand(p, \mathcal{D}, \theta)$ procedure uses a "follow join" to calculate the total pairs of an event in E' and an event of E that satisfies the *follow* predicate and returns this number. If the computed ratio is above the user-given threshold θ, a new node with the current event type is added as a child of p, and the newly created node is recursively expanded. Well-known algorithms, for example, plane sweep [18], space partition [19], and tree matching [20] may be used for the *follow join* in order to compute density ratios and tail event sets. We choose the plane sweep algorithm for STS-Miner due to its low overhead and simplicity. The tail event set will be cleared to save space after a node is fully expanded at the end. The details of the depth-first expansion procedure are shown as above.

depthFirstExpand$(\mathbf{p}, \mathcal{D}, \theta)$
p: a node representing a sequence that starts from the root of a pattern tree;
$\mathcal{D}$: a set of K event types and their event sets $\mathcal{D} = \{\{f_1, f_1.\mathcal{E}\}, \{f_2, f_2.\mathcal{E}\}, \ldots, \{f_K, f_K.\mathcal{E}\}\}$;
θ: the sequence index threshold

1: **for** each event type f in $\mathcal{D}$ **do**
2: $joinSize \leftarrow join(p.tailEventSet \rightarrow f.\mathcal{E})$;
3: $tailEventSet \leftarrow findTailEventSet (p.tailEventSet \rightarrow f.\mathcal{E})$
4: $ratio \leftarrow calculateDensityRatio (joinSize, size(tailEventSet))$;
5:
6: **if** $(ratio \geq \theta)$ **then**
7: $newP \leftarrow expandSequence (p, f, tailEventSet, joinSize, ratio)$
8: /* Expand sequence p by appending f */
9: $depthFirstExpand(newP, \mathcal{D}, \theta)$;
10: **end if**
11: **end for**
12: $setEmpty(p.tailEventSet)$;

The procedure

$$expandSequence(p, f, eventSet, joinSize, ratio),$$

as shown in the following, creates a child for a node p if it does not have a child with the event type f yet. Otherwise, it updates the counters and arguments of the tail event set of the existing child.

expandSequence$(\mathbf{p}, \mathbf{f}, \mathbf{tailEventSet}, \mathbf{joinSize}, \mathbf{ratio})$
p: a node representing a sequence that starts from the root of a pattern tree;
f: an event type;
$eventSet$: tail event set of $p.tailEventSet \rightarrow f.\mathcal{E})$;
$joinSize$: joinSize of $join(p.tailEventSet \rightarrow f.\mathcal{E})$;
$ratio$: density ratio of $join(p.tailEventSet \rightarrow f.\mathcal{E})$;

1: **if** (p has a child cp with event type f) **then**
2: $cp.joinSize \leftarrow cp.joinSize + joinSize$;
3: $cp.tailEventSetSize \leftarrow cp.tailEventSetSize + size(tailEventSet)$;
4: $cp.tailEventSet \leftarrow cp.tailEventSet \cup tailEventSet$;
5: **return** cp;
6: **else**
7: $newP \leftarrow new(patternTreeNode)$;

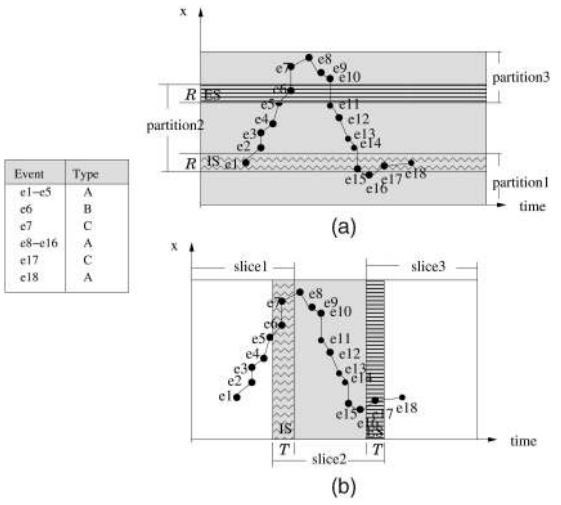


Fig. 5. (a) Spatial partition: an event sequence of a sequential pattern may visit a spatial partition multiple times. (b) Temporal slicing: an event sequence of a sequential pattern only visits a temporal slice once.

8: $newP.eventType \leftarrow f$;
9: $cp.tailEventSetSize \leftarrow size(tailEventSet)$;
10: $newP.joinSize \leftarrow joinSize$;
11: $newP.tailEventSet \leftarrow eventSet$;
12: $cp.densityRatio \leftarrow ratio$;
13: add $newP$ to p's child list;
14: **return** newP;
15: **end if**

3.2 Slicing-STS-Miner

When the spatio-temporal database is small and the mining process can be carried out in the main memory, the naive STS-Miner works fine. However, when a spatio-temporal data set is large and cannot be fit into the main memory, STS-Miner could be very expensive due to the excessive number of database scans.

In spatial databases, spatial partitioning is frequently used to process both pairwise and multiway spatial joins [18], [19], [21]. However, in spatio-temporal sequential pattern mining, an event sequence of a long spatio-temporal sequential pattern could spread across multiple spatial partitions and visit a partition more than once. For example, in Fig. 5a, an event sequence $e_1 \rightarrow e_2 \ldots \rightarrow e_{18}$ is shown. When we partition the space spatially, we have three partitions: $partition_1$, $partition_2$, and $partition_3$. Furthermore, we maintain a small overlapping area between consecutive partitions, for example, IS and ES in Fig. 5a, with their sizes decided by the spatial join predicate, to reclaim join results across arbitrary partition boundaries. The idea of spatial partitioning-based algorithms is that we can process partitions in some order, for example, $partition_1$, $partition_2$, and $partition_3$, without revisiting the processed partitions. However, because the event sequence $e_1 \rightarrow e_2 \ldots \rightarrow e_{18}$ extends from $partition_1$ to $partition_2$ to $partition_3$ and then back to $partition_2$ and, finally, to $partition_1$, it is not possible to process the three spatial partitions in some order and discard each partition once processed.

However, we observe that unlike space, which can extend the neighborhood in both directions (for example, up and down, as shown in Fig. 5a), time has a unidirectional property (for example, right direction, as shown in Fig. 5b). This leads us to a temporal slicing-based algorithm, namely, the **Slicing-STS-Miner**, that deals with temporal slices in a piece-meal fashion and guarantees one database scan in the processing of the slices. The Slicing-STS-Miner algorithm has three phases: the hashing phase, the mining and merging phase, and the pruning phase. The hashing phase slices the database according to time and hashes events into slices. Then, we mine sequential patterns from each slice and, at the same time, merge with the patterns found in the previous slice through updating a common pattern tree. The last phase prunes the pattern tree by using the given sequence index threshold. The details of the algorithms are shown in Algorithm 1 as follows:

Algorithm 1: Slicing-STS-Miner:
Input:
A set of K event types and their event sets $\mathcal{D} = \{\{f_1, f_1.\mathcal{E}\}, \{f_2, f_2.\mathcal{E}\}, \ldots, \{f_K, f_K.\mathcal{E}\}\}$;
A user-specified minimum sequence index threshold θ;
Size of each time slice $\mathcal{S}_t$;
Output: A significant pattern tree patternTree
Variables:
$oneSlice$: a time slice of events, which can fit in the main memory;
$crossingQ$: a queue maintaining pointers to the nodes of the pattern tree with crossing tail event from the last slice
$newCrossingQ$: a queue maintaining pointers to the nodes of the pattern tree with crossing tail event for the current slice

1: $\mathcal{D}' \leftarrow hashing(\mathcal{D}, \mathcal{S}_t)$
2: **while** $(moreslices\ left\ in\ \mathcal{D})$ **do**
3: $oneSlice \leftarrow takeOneSlice(\mathcal{D}')$;
4: **if** $(thefirstslice)$ **then**
5: **for** (each event type f in $oneSlice$) **do**
6: /* Create a child for patternTree by appending f */
7: $expandSequence(patternTree, f, f.\mathcal{E}, 0, 1)$;
8: **end for**
9: **else**
10: /*Expand the nodes with crossing tail event(s) in $crossingQ$ and generate a new crossing event queue $newCrossingQ$*/
11: **while** $(q \leftarrow dequeue(crossingQ))$
12: $q.tailEventSet \leftarrow q.crossingTailEventSet$;
13: $depthFirstExpand^*(q, oneSlice, newCrossingQ)$;
14: **end while**
15: /*Update the pattern tree using the current slice*/
16: **for** (each child p of patternTree) **do**
17: $\mathcal{E} \leftarrow$ event set of event type $p.eventType$ in oneSlice;
18: $p.tailEventSet \leftarrow \mathcal{E}$;
19: $depthFirstExpand^*(p, oneSlice, newCrossingQ)$;

20: $crossingQueue \leftarrow newCrossingQ$;
21: $setEmpty(newCrossingQ)$;
22: **end for**
23: **end if**
24: **end while**
25: /*recursively compute the sequence index and prune the pattern tree*/
26: $calculateSeqIndex(patternTree, \theta)$;
27: **return** $patternTree$;

In the hashing phase (line 1), we first divide the time dimension into overlapping slices. Every two consecutive slices overlap by $\mathcal{T}$, where $\mathcal{T}$ is the given time interval for the *follow* relationship. We *hash* each event into slices by its time stamp. One event could be hashed into two slices if its time stamp is in the overlapping area of two slices.

Fig. 5b shows a spatio-temporal space divided into three slices according to the time dimension. Events e_6 and e_7 are hashed into both $slice_1$ and $slice_2$. Event e_{17} is hashed into $slice_2$ and $slice_3$. Events e_1, e_2, e_3, e_4, and e_5 are hashed into $slice_1$ only. Events $e_8, e_9, \ldots, e_{16}$ are hashed into $slice_2$ only. Note that the hashing is based on the time stamps of the events only. We choose the length of the slices as large as possible, as long as we can process the events in one slice in the main memory.

In the mining and merging phase (lines 2-24), we process the slices in a time-increasing order while updating a single pattern tree similar to that shown in Fig. 3. This is possible because of both the *follow* relationship's and time's unidirectional property, which guarantees that the event sequences extend along the time line and do not travel reversely. For the first slice (lines 4-8), we simply create the initial pattern tree consisting of all the singleton sequences.

When processing slices other than the first one, we face two challenges in the piece-meal processing: 1) *Duplicate sequences in overlapping areas of two consecutive slices*. In Fig. 5b, e_6 is of event type B, and e_7 is of event type C, as shown. $e_6 \rightarrow e_7$ will be counted as an event sequence for $B \rightarrow C$ when processing slice $slice_1$ and, again, in processing slice $slice_2$, resulting in a higher sequence index than the actual one. In fact, all event sequences that are located entirely in the overlapping areas of two overlapping slices will be over counted. 2) *Sequences broken by boundaries of consecutive slices*. In Fig. 5b, event sequences extending beyond the boundary of one slice will be lost. For example, event sequences $e_5 \rightarrow e_6 \ldots \rightarrow e_8$ and $e_1 \rightarrow e_2 \ldots \rightarrow e_{17}$ will not be found in a simple piece-meal processing fashion.

One way of addressing the duplicate and broken sequence problems is to keep track of all of the event sequences that visit the overlapping areas of two consecutive slices for duplication elimination and broken sequence recovery. However, this approach is not only space and time consuming but also involves complex sequence alignment process in order to identify all the sequences without introducing duplication. If we simply match the last event of an event sequence $\overrightarrow{E}$ with the first event of another event sequence $\overrightarrow{E}'$ to produce a longer sequence, we may generate the combined sequence multiple times. For example, in Fig. 5, $e_5 \rightarrow e_6 \rightarrow e_7$ and $e_5 \rightarrow e_6$ are two events sequences that end up in the overlapping area of $slice_1$ and $slice_2$ after we process $slice_1$. $e_7 \rightarrow e_8$ and $e_6 \rightarrow e_7 \rightarrow e_8$ are two events sequences after we process $slice_2$. By simply matching the tails of the first two sequences and the heads of the second two sequences, we will generate $e_5 \rightarrow e_6 \rightarrow e_7 \rightarrow e_8$ twice.

Thus, we propose an algorithm based on the concept of *crossing tail event set* to handle both the duplicate-sequence and broken-sequence problems simultaneously without dealing with the demanding space overhead and the complicated duplicate elimination and sequence alignment problems.

Definition 8 (Crossing Tail Event Set). *An event e is called a crossing tail event of a (k)-sequence $\overrightarrow{S}$ if 1) $e \in tail\_event\_set(\overrightarrow{S})$, 2) e is located in the overlapping area of two consecutive slices $slice_i$, $slice_{i+1}$, and 3) when $k > 1$, $\exists e' \in tailEventSet(\overrightarrow{S}[1 : k-1])$ such that e' is in $slice_i$ but is not located in the overlapping area and $e' \rightarrow e$. The set of all the crossing tail event of a (k)-sequence $\overrightarrow{S}$ is called the crossing tail event set of $\overrightarrow{S}$ and is denoted by $crossingTailEventSet(\overrightarrow{S})$.*

In Fig. 5b, e_6 is a crossing tail event of sequence patterns $A \rightarrow B$, $A \rightarrow B$, and $A \rightarrow A \rightarrow A \ldots \rightarrow B$. Similarly, e_{17} is a crossing tail event of sequence patterns $A \rightarrow C$, $A \rightarrow A \rightarrow C$, and $A \rightarrow C \rightarrow C \ldots \rightarrow C$, among many others. However, e_7 is not a crossing tail event of any sequence in the figure, since it does not follow any event that is in $slice_1$ and is not located in the overlapping area. We propose a modified depth-first search procedure that keeps track of the nodes with crossing tail events and maintains them into a queue $crossingQ$.

depthFirstExpand$^*(p, \mathcal{D}, \mathbf{crossingQ})$
p: a node representing a sequence that starts from the root of a pattern tree;
$\mathcal{D}$: A set of K event types and their event sets $\mathcal{D} = \{\{f_1, f_1.\mathcal{E}\}, \{f_2, f_2.\mathcal{E}\}, \ldots, \{f_K, f_K.\mathcal{E}\}\}$;
$CrossingQ$: a queue maintaining points to the nodes of the pattern tree with crossing tail event

1: **for** (each event type f in $\mathcal{D}$) **do**
2: $joinSize \leftarrow join((p.tailEventSet) \rightarrow f.\mathcal{E})$;
3: $crossingTailEventSet \leftarrow findCrossingTailEventSet(p.tailEventSet \rightarrow f.\mathcal{E})$;
4: **if** $(crossingTailEventSet \neq empty)$ **then**
5: $enqueue(crossingQ, p)$;
6: **end if**
7: /* Expand sequence p by appending f */
8: $newP \leftarrow expandSequence(p, f, joinSize, tailEventSet, 0)$;
9: $newP.crossingTailEventSet \leftarrow crossingTailEventSet$;
10: $depthFirstExpand^*(newP, \mathcal{D})$;
11: **end for**
12: $setEmpty(p.tailEventSet)$;

To solve the broken sequence problem, the modified depth-first search algorithm creates a queue of sequences that have at least one crossing tail event. The crossing event set is maintained within a sequence represented by a node

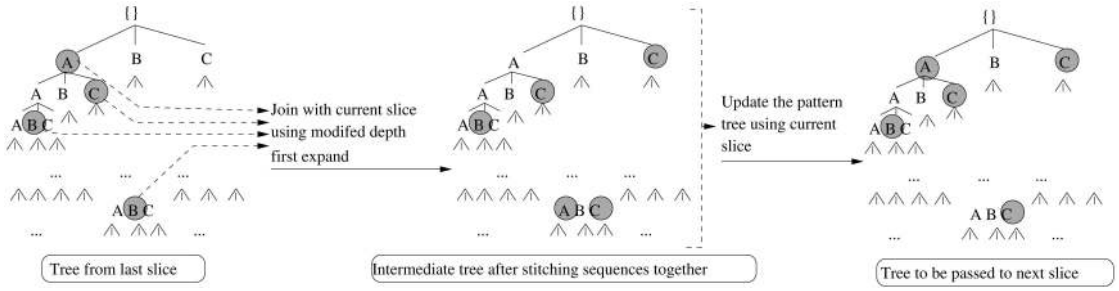


Fig. 6. Illustration of pattern trees from the last slice going through duplicate elimination/broken-event-sequence recovery and then update by the current slice. Nodes with crossing tail events are shaded.

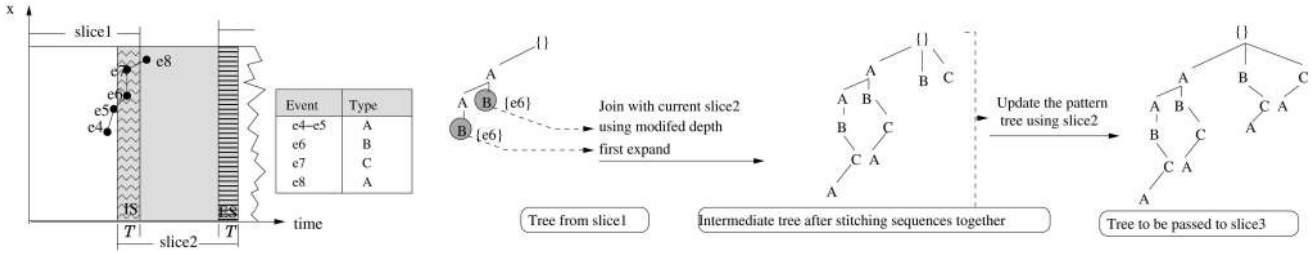


Fig. 7. A sample data set to illustrate the pattern tree from the last slice going through duplicate elimination/broken event sequence recovery and then update by the current slice.

in the pattern tree. In Fig. 6a, the shaded nodes of the pattern tree represent the sequences with a crossing tail event after processing slice $slice_1$ and are put into a queue $crossingQ$. To recover the broken sequences, the nodes with crossing tail event in q are expanded first by using the modified depth-first search algorithm on the next slice, for example, $slice_2$. As a result, a new queue of sequences $newCrossingQ$ that have at least one crossing tail event for the current slice will be created and to be passed to the next slice. Then, the whole pattern tree is updated using the modified depth-first search procedure and the current slice. The $newCrossingQ$ may be extended further in the next slice.

Fig. 7 shows a smaller sample data set and the states of the pattern tree after the Slicing-STS-Miner processes $slice_1$ and $slice_2$. After we process the $slice_1$, two nodes (shaded) have crossing tail events. Thus, these two nodes keep track of the crossing tail events. The tail event sets for other nodes will be empty, since they are not needed any more. Then, the two shaded nodes are expanded using events in $slice_2$. Since there are no crossing events between $slice_2$ and the next slice, there are no shaded nodes in the pattern tree after this step. Then, the tree is updated using events in $slice_2$. Again, there are no shaded nodes, since there are no nodes with crossing tail event(s).

Note that when we process a slice by using the modified depth-first search, we did not use the sequence index threshold, since we have to wait until we have processed all the slices and then do the pruning.

After we have processed all the slices, we have a pattern tree, where each pattern's sequence index could be easily calculated using the numbers associated with the nodes and branches. We prune the pattern tree by using the minimum sequence index threshold θ (line 26) of Algorithm 1.

3.3 Cost Analysis of the Proposed Algorithms

Lemma 3. *Procedure depthFirstExpand is complete and correct and terminates in a finite number of iterations.*

Lemma 4. *Slicing-STS-Miner is complete and correct.*

We omit the proofs of the above two lemmas due to space constraints.

We present the algebraic cost models of the algorithms on small data sets suitable for in-memory processing first, followed by the algebraic cost models of the algorithms on large data sets.

3.3.1 Cost Models for In-Memory Processing

For in-memory processing, the I/O costs are the same for all the algorithms and are equal to the costs of loading the data sets into the memory and writing out the results. In order to expand any subsequence $\overrightarrow{S}$ by one more event type f, we need to perform a spatio-temporal join of the tail event set of $\overrightarrow{S}$ with the event set of f. Letting the average size of an event set of an event type be F_i, then the average size of a tail event set can be estimated by αF_i, where α is a nonnegative number and is not larger than 1. When both the tail event sets and the event sets participating in the spatio-temporal join are ordered according to time, the cost of one plane-sweep-based join algorithm [18] is given as follows, assuming that each event is followed by a close-to-constant number of other events:

$$cost_{sweep} = O(F_i + \alpha F_i) = O(F_i).$$

The tail event set of the expanded sequence, as a result of the plan sweep-based join, will be naturally ordered according to time after the join and is ready to be passed to the next expansion phase.

Let p_n be the number of maximal sequential patterns and p_s be the mean length of the maximal sequential patterns. The total number of significant patterns can be estimated by

$N_{sig} = p_n \times 2^{p_s} \times \rho$, where ρ is less than 1 to indicate the overlapping degree of the sequences.

To find all the significant sequences, at least N_{sig} number of spatial joins need to be performed, assuming that the unsuccessful joins are proportional to the successful joins that produce significant sequences. The total cost for the STS-Miner can be estimated as

$$cost^m_{STS-Miner} = cost_{sweep} \times N_{sig} = O(F_i \times p_n \times 2^{p_s} \times \rho).$$

Even when all the data can be fit into the main memory, a full-blown pattern tree is generated first by using the sequence index threshold of 0 for the STS-Slicing-Miner. Then, the tree is pruned using the user-given sequence index threshold. The cost of STS-Slicing-Miner can be calculated similarly, except that the total number of patterns found by the algorithm before pruning can be larger than those found by STS-Miner. Furthermore, we need to include the cost of pruning the tree. The pruning cost can be estimated as the cost of traversing the final pattern tree:

$$cost_{pruning} = O(p'_n \times 2^{p'_s} \times \rho),$$

where $p'_n \geq p_n$, and $p'_s \geq p_s$. In addition, the cost for STS-Slicing-Miner is

$$\begin{aligned} cost^m_{STS-Slicing-Miner} &= cost_{sweep} \times N_{sig} + cost_{pruning} \\ &= O(F_i \times p'_n \times 2^{p'_s} \times \rho). \end{aligned}$$

Note that the hashing/slicing cost is not included, because the data set is small and can be treated as one slice and can be fit into memory.

3.3.2 Cost Models for Large Data Sets

For a large spatio-temporal data set, the assumption of being able to load the whole data set into the memory is generally false. Assuming, aside from maintaining, the sequence tree and performing basic operations of the algorithms, the system can load M events into the main memory, we now investigate the relationship between the memory size and the performance of the algorithms. For a data set $\mathcal{D}$, the cost of the two algorithms now can be represented as

$$\begin{aligned} cost^d_{STS-Miner} &= cost^m_{STS-Miner} + n_{STS-Miner} \times cost_{load\mathcal{D}}, \\ cost^d_{Slicing-STS-Miner} &= cost^m_{STS-Slicing-Miner} \\ &\quad + n_{STS-Slicing-Miner} \times cost_{load\mathcal{D}}, \end{aligned}$$

where $cost_{load\mathcal{D}}$ represents the cost of loading D into the main memory, and $n_{STS-Miner}/n_{STS-Slicing-Miner}$ denotes the number of times that STS-Miner/Slicing-STS-Miner loads the whole data set $\mathcal{D}$ into the main memory.

When the system can load in the events of one event type every time, that is, $M \sim F_i$, then $n_{STS-Miner}$ is the average number of times that an event type f participates in a pattern, since the whole event set for an event type f needs to be loaded every time a pattern is expanded to include f.

Because choosing the right slicing size and the total number of slices for Slicing-STS-Miner are very important, we provide the guidelines here. Let M be the memory size available and M_p represent the estimated memory needed to maintain the pattern tree and perform basic operations such as joins. Then, the memory left for loading a data set will be $M - M_p$. Let the size of each slice be s_{slice} and the size of the overlapping area of two consecutive slices be $s_{overlap}$. Then, $s_{slice} + s_{overlap} = M - M_p$, and $s_{slice} = M - M_p - s_{overlap}$. Letting the size of the data set $\mathcal{D}$ be $S_{\mathcal{D}}$, data set $\mathcal{D}$ should be partitioned into n_{slice} slices so that

$$n_{slice} \times (M - M_p) = S_{\mathcal{D}} + (n_{slice} - 1) \times s_{overlap}.$$

Thus, we get $n_{slice} = \lceil \frac{S_{\mathcal{D}} - s_{overlap}}{M - M_p - s_{overlap}} \rceil$.

For Slicing-STS-Miner, the consecutive slices overlap to help in recovering sequences across slice boundaries. Some of the events in the overlapping areas are duplicated and stored in both slices with an average size of $s_{overlap}$. The increase in the percentage of the hashed data set $\mathcal{D}'$ from $\mathcal{D}$ is $p_{increase} \approx \frac{s_{overlap}}{s_{slice}} \approx \frac{s_{overlap}}{M - M_p - s_{overlap}}$. The data set $\mathcal{D}$ will be hashed first, resulting in one database scan for reading and $1 + p_{increase}$ database scan in writing. Then, the hashed data set will be loaded slice by slice and only once during the mining and merging phase, resulting in a $1 + p_{increase}$ database scan. In the pruning phase, no database scan is needed. The pattern tree is kept in the memory and is pruned only after all the slices are processed. In this case, Slicing-STS-Miner requires $3 + 2 \times p_{increase}$ database scans in total. Thus, we have the following:

Lemma 5. *The number of times that Slicing-STS-Miner scans the database is* $3 + 2 \times \frac{s_{overlap}}{M - M_p - s_{overlap}}$.

Then, when $p_{increase}$ is small, for example, 1 percent and $s_{overlap} \leq 0.09 \times (M - M_p)$, the duplication of the events in the overlapping area is negligible, resulting in three database scans.

TABLE 2
Parameters of Data Generator

Parameter	Definition
p_n	number of max sequential patterns
p_s	mean length of max sequential patterns
N_f	number of event types participating in a pattern
N_n	number of noise events
N_i	average number of event sequence for each sequential pattern
T_i	total number of events
$\mathcal{R}$	maximal distance two event related
$\mathcal{T}$	maximal time interval two events related
s_{size}	the size of each slice
s_i	sequence index threshold

4 Experimental Design and Results

We conducted an extensive performance study on both synthetic and real data sets. All the experiments are performed on a Pentium 4 3.2-GHz PC machine with 1-Gbyte main memory, running Fedora Linux. All algorithms are implemented using C++.

4.1 Results on Synthetic Data Sets

Synthetic data are generated using a data generator similar to the one by Agarwal and Srikant [22], along with an extension to produce spatial and temporal data sets. The important parameters are explained in Table 2.

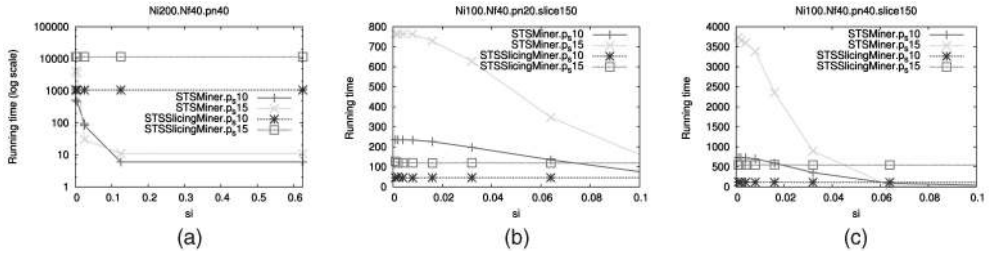


Fig. 8. Performance with respect to the sequence index. (a) In memory. (b) Disk. (c) Disk.

We generate an average of N_i event sequences for each sequential pattern and the same number of noise events as well. This will result in about $T_i = N_i \times p_n \times p_s \times 2$ events. For the event sequence of a maximal sequential pattern, we choose from a set of maximal sequential pattern of size p_n, with a probability described in the next paragraph. Then, we choose a spatio-temporal framework of size $T \times D \times D$, generate an event with a probability described in the next graph for each event type in the chosen maximal sequential pattern, and put it uniformly into the spatio-temporal neighborhood of the previous event in the event sequence (the first event is randomly put into the space). If the event generated is a random event, we choose an event type with equal probability from the event type set of size N_f and put an event of that event type uniformly into the space.

The maximal potential sequential pattern set P is pregenerated as follows: A sequential pattern in P is generated by first choosing its size from a Poisson distribution with mean p_s. Event types in the sequential pattern are chosen randomly. Each sequential pattern has a weight corresponding to the probability that this sequential pattern will be chosen. The weight is chosen from an exponential distribution, with the unit mean normalized by the sum of the weights. For each event type, there is a probability that an event will be generated. This probability is chosen from an exponentially distributed random variable, with the unit mean normalized by the maximal of the weights of each event type.

In our experiments, the space and time framework is set to $1,200 \times 1,000 \times 1,000$. When not specified, the maximal distance $\mathcal{R}$ is set to 10, and the maximal time interval $\mathcal{T}$ that two events could be related is set to 10. The number of event types N_f participating in a pattern is set to 100. We use the notation $P_n30.P_s5.N_f100.N_i10k.si0.1$ to denote that the number of the maximal sequential patterns is 30, the mean size of the maximal sequential patterns is 5, the number of event types participating in a pattern is 100, the total number of events is 10,000, and the minimum sequence index threshold is 0.1.

We conducted two sets of experiments for each parameter: in-memory based and disk based. For in-memory-based experiments, the buffer size is set to the maximal so that all events can be loaded into the memory. For disk-based experiments, the buffer size is set to 10,000 events. As a result, we force the algorithms to read in up to 10,000 events each time. Buffer management is implemented in a circular-array fashion.

4.1.1 Effect of Sequence Index Thresholds

We fixed other parameters and varied the sequence index thresholds to investigate the effect of the sequence index thresholds on the two algorithms. Fig. 8 shows the performance of the two algorithms on a data set when the maximal sequential patterns (p_n) is 20 or 40, the mean size of the maximal sequential patterns (p_s) is 10 and 15, and the average number of event sequences of a pattern (N_i) is 100 or 200.

For in-memory experiments, as shown in Fig. 8a, STS-Miner is two or three orders of magnitude faster than the STS-Slicing-Miner algorithm for large sequence indices (greater than 0.1). The closer the sequence index to 0, the closer the performance of the two algorithms. For the sequence index of 0, STS-Miner still performs two to three times better than STS-Slicing-Miner. This is expected, as STS-Slicing-Miner has to perform the extra work of connecting sequences across slicing boundaries.

For disk-based experiments, STS-Slicing-Miner is up to an order of magnitude faster than STS-Miner. The performance advantage decreases as the sequence index threshold increases. With the sequence index decreasing, the STS-Miner runs slower, whereas the performance of the Slicing-STS-Miner remains the same. We will investigate the selection of slice size further in Section 4.1.5 shortly.

4.1.2 Effect of the Average Number of Event Sequences for Each Pattern

Fig. 9 shows the performance of the two algorithms with respect to the average number of event sequences.

For the in-memory-based experiments, the number of maximal sequential patterns p_n is 20, the mean size of the maximal sequential patterns p_s is 10 or 15, and the sequence index threshold s_i is 0.001. With the average number of event sequences for each pattern increasing, the runtime also increases for both algorithms. The performance of both algorithms are slightly super linear with respect to the average number of event sequences for each pattern. STS-Miner outperforms STS-Slicing-Miner by a factor of around 4 for a pattern size of 15 and by a factor of around 3 for a pattern size of 10. STS-Miner is close to linear with respect to the total number of events, whereas STS-Slicing-Miner is slightly super linear with respect to the total number of events, as shown in Fig. 9. This observation is consistent with the memory-based cost models, where the costs are linear with respect to p_n.

For the disk-based experiments, STS-Slicing-Miner is linear (or slightly superlinear) with respect to both the

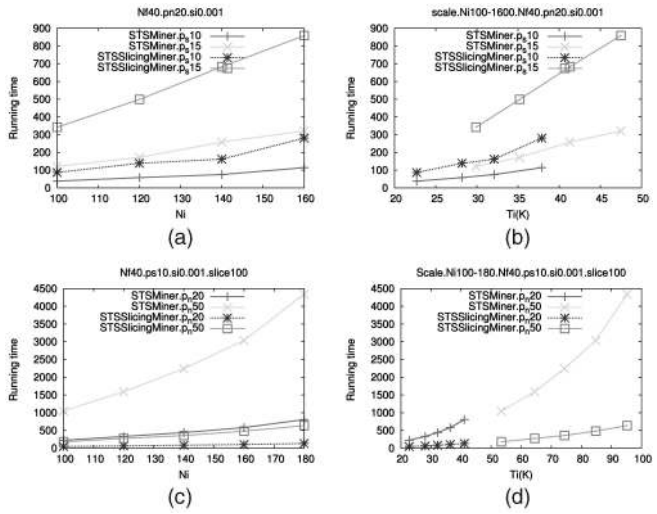


Fig. 9. Performance with respect to the average number of event sequences for a pattern. (a) In memory. (b) In memory. (c) Disk. (d) Disk.

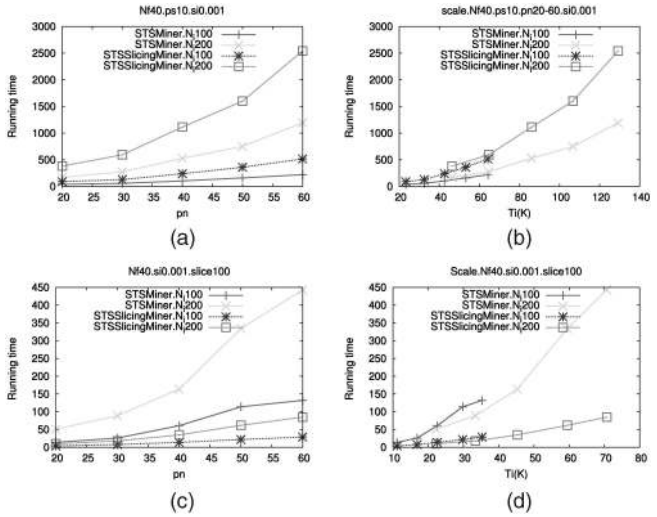


Fig. 11. Performance with respect to the number of patterns. (a) In memory. (b) In memory. (c) Disk. (d) Disk.

average number of event sequences and the total number of events, which is consistent with the algebraic cost model. STS-Miner is exponential with respect to both the average number of event sequences and the total number of events. STS-Slicing-Miner is almost an order of magnitude faster than STS-Miner. This means that Slicing-STS-Miner is much more scalable than STS-Miner for large data sets with many events.

4.1.3 Effect of the Average Pattern Size

Fig. 10 shows the performance of the algorithms with respect to the average pattern size. For the in-memory-based experiments, both algorithms are exponential. This is mainly due to the fact that the number of patterns found increases exponentially with respect to the average pattern size, as indicated in the algebraic cost model. The growth rate of STS-Slicing-Miner is much faster than that of STS-Miner when the whole data set can fit into the memory.

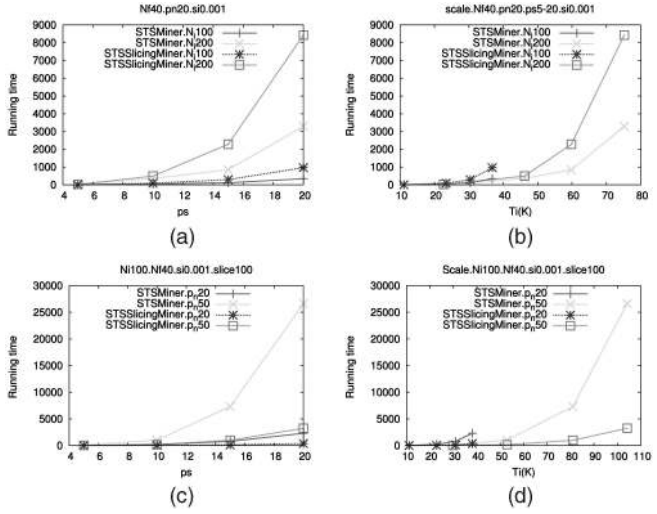


Fig. 10. Performance with respect to the average pattern size. (a) In memory. (b) In memory. (c) Disk. (d) Disk.

For the disk-based experiments, STS-Slicing-Miner is close to linear with respect to both the pattern size and the total number of events for small maximal sequential patterns (for example, $p_n = 20$), whereas STS-Miner is exponential. STS-Slicing-Miner is up to an order of magnitude faster than the STS-Miner for large pattern sizes.

4.1.4 Effect of the Number of Patterns

Fig. 11 shows the performance of the algorithms with respect to the number of patterns in the data sets for both in-memory processing and disk-based processing. For the in-memory-based experiments, STS-Slicing-Miner is about two times slower than the STS-Miner algorithm, as shown in Fig. 11a, for various numbers of patterns. When it comes to limited memories, where the number of I/Os increases, STS-Slicing-Miner is an order of magnitude faster than STS-Miner, as shown in Fig. 11b.

4.1.5 Effect of Slicing Size on Slicing-STS-Miner

Fig. 12a shows the performance of Slicing-STS-Miner, with the slice size increasing. For this data set, the maximal sequential pattern p_n is 30, the mean size of the maximal sequential patterns p_s is 5, the number of event types participating in a pattern N_f is 60, the total number of events T_i is 20,000, and the sequence index threshold s_i is 1. Although the sequence index threshold is chosen to be 1, Slicing-STS-Miner is not sensitive to the sequence index threshold. The performance of Slicing-STS-Miner increases

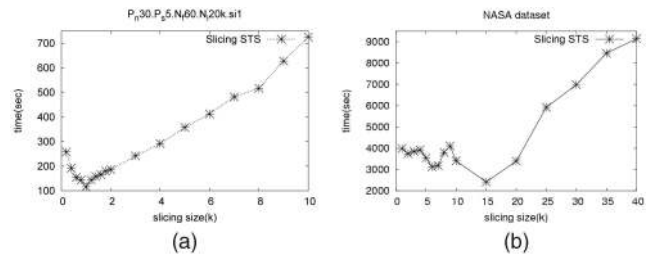


Fig. 12. (a) Performance with respect to the slicing size. (b) Performance with respect to the slicing size on real data sets.

first monotonically before reaching the optimal and then decreases monotonically. The optimal slicing size is 1,000. The reason for the performance decrease for small slicing sizes is that when the data set is sliced into too small pieces, the overheads in handling interior and exterior sets eclipse the potential savings of reduced I/O. When the slicing size is too large, each slice may not fit into main memory, resulting in busy paging swap and, thus, horrendous performance.

4.2 Results on Real Data Sets

We conducted an experimental study by using a real-world climate data set [23]. The NASA Earth observation systems generate a large sequence of global snapshots of the Earth. The data includes the following climate variables:

1. NPP,
2. temperature,
3. precipitation,
4. solar radiation, and
5. evaporation.

The NPP is the net photosynthetic accumulation of carbon by plants. Keeping track of the NPP is important, because it includes the food source of humans and all other organisms and, thus, sudden changes in the NPP of a region can have a direct impact on the regional ecology. An abnormal event is captured if its absolute value of the statistical normal test, which is typically by subtracting the mean and dividing by the standard deviation of the time series, is beyond two in this data set. For example, if the average temperature in January is $20F$ with a standard deviation of $2F$ in a location, the January with $10F$ is considered as abnormally cold, and the January with $30F$ is consider as abnormally warm in the same location.

The event types include high and low events for those five climate variables. There are roughly 67,000 spatial locations in the data, and monthly abnormally high and low events for those five climate variables were captured during the 1982 to 1993 time window at those locations. The data are in the format of longitude, latitude, time stamp, and event type. We applied our approach on this data set to mine sequential patterns from this spatio-temporal data set, and different sequence index thresholds were used in the study. The spatial neighborhood is chosen as $\mathcal{R} = 5$ half degrees, and the time interval is $\mathcal{T} = 5$ months. The main goal of this study is to identify some potential interesting patterns, which might reveal possible intrinsic relationships among climate variables and provide candidate hypotheses for further investigation by Earth scientists.

As shown in Fig. 12b, the performance of Slicing-STS-Miner has slight fluctuations in the early part of the curve. The overall performance of the algorithm reaches its best when each slice contains approximately 15,000 events and, then, the performance decreases. The reasons are similar as explained in Section 4.1.5 and the algebraic cost model.

In the experimental results, we saw many self-correlated sequential patterns, for example, one low-temperature event type is followed by another low-temperature event type, and these kinds of patterns might be trivial and are thus not interesting for this application. After removing those self-correlated patterns, the patterns found are shown in Table 3. The sequential pattern of a high-temperature event type followed by a high-evaporation event type consistently showed up when different sequence index thresholds were used. This pattern indicates that an abnormally higher temperature could possibly lead to abnormally higher evaporation. We also found that a low-evaporation event type is followed by a high-temperature event type, and even a longer chain of low evaporation, high temperature, and high evaporation is shown in the results. Those results are interesting, since they might indicate some potentially intrinsic interactions among low-evaporation, high-temperature, and high-evaporation climate events, which are subject to further investigations by our Earth science colleagues.

TABLE 3
Sequential Patterns Found in the Climate Data on Various Sequence Index Thresholds

θ	Sequential Patterns
14	High Temperature → High Evaporation
12	High Temperature → High Evaporation
10	High Temperature → High Evaporation
8	High Temperature → High Evaporation
6	High Temperature → High Evaporation
4	High Temperature → High Evaporation Low Evaporation → High Temperature Low Evaporation → High Temperature → High Evaporation

5 Conclusion and Future Work

In this paper, we formally defined the problem of mining spatio-temporal sequential patterns. We proposed a sequence index as the spatial significance measure for spatio-temporal sequences and established the statistical interpretation by using spatial statistics. We proposed two algorithms for sequential pattern mining on spatio-temporal data. The proposed algorithms tackled the algorithmic challenges by using the sequence index, which did not guarantee the downward closure property. We conducted extensive performance evaluations for the proposed algorithms, using both real and synthetic data sets. Further research may include studying the effects of using different variable spatial neighborhood and time intervals for one sequence instead of the constant neighborhood and time intervals that we have assumed here.

Acknowledgments

The authors are particularly grateful to Dr. Vipin Kumar and Mr. Michael Steinbach from the University of Minnesota for their support in providing the climate data for our experimental evaluations.

References

[1] M. Koubarakis, T.K. Sellis, A.U. Frank, S. Grumbach, R.H. Güting, C.S. Jensen, N.A. Lorentzos, Y. Manolopoulos, E. Nardelli, B. Pernici, H.-J. Schek, M. Scholl, B. Theodoulidis, and N. Tryfona, *Spatio-Temporal Databases: The CHOROCHRONOS Approach.* Springer, 2003.

[2] J. Roddick and M. Spiliopoulou, "A Bibliography of Temporal, Spatial and Spatio-Temporal Data Mining Research," *ACM SIGKDD Explorations,* http://kdm.first.flinders.edu.au/IDM/STDMBib.html, 1999.

[3] P. Zhang, M. Steinbach, V. Kumar, S. Shekhar, P. Tan, S. Klooster, and C. Potter, "Discovery of Patterns of Earth Science Data Using Data Mining," *Next Generation of Data Mining Applications,* 2004.
[4] Y. Huang, L. Zhang, and P. Zhang, "Finding Sequential Patterns from a Massive Number of Spatio-Temporal Events," *Proc. Sixth SIAM Int'l Conf. Data Mining (SDM '06),* 2006.
[5] C. for Disease Control and P. (CDC), CDC West Nile Virus Homepage, http://www.cdc.gov/ncidod/dvbid/westnile, 2007.
[6] R. Srikant and R. Agrawal, "Mining Sequential Patterns: Generalizations and Performance Improvements," *Proc. Fifth Int'l Conf. Extending Database Technology (EDBT '96),* pp. 3-17, 1996.
[7] M.J. Zaki, "SPADE: An Efficient Algorithm for Mining Frequent Sequences," *Machine Learning,* vol. 42, no. 1/2, pp. 31-60, 2001.
[8] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach," *IEEE Trans. Knowledge and Data Eng.,* vol. 16, no. 11, pp. 1424-1440, Nov. 2004.
[9] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu, "Freespan: Frequent Pattern-Projected Sequential Pattern Mining," *Proc. Sixth ACM SIGKDD,* 2000.
[10] R. Agrawal and R. Srikant, "Mining Sequential Patterns," *Proc. IEEE 11th Int'l Conf. Data Eng. (ICDE '95),* pp. 3-14, 1995.
[11] Y. Huang, S. Shekhar, and H. Xiong, "Discovering Colocation Patterns from Spatial Datasets: A General Approach," *IEEE Trans. Knowledge and Data Eng.,* vol. 16, no. 12, Dec. 2004.
[12] B. Ozden, S. Ramaswamy, and A. Silberschatz, "Cyclic Association Rules," *Proc. 14th IEEE Int'l Conf. Data Eng. (ICDE '98),* 1998.
[13] J. Han, G. Dong, and Y. Yin, "Efficient Mining of Partial Periodic Patterns in Time Series Database," *Proc. 15th IEEE Int'l Conf. Data Eng. (ICDE '99),* 1999.
[14] N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, and D.W.L. Cheung, "Mining, Indexing, and Querying Historical Spatiotemporal Data," *Proc. 10th ACM SIGKDD,* 2004.
[15] H. Cao, D.W. Cheung, and N. Mamoulis, "Discovering Partial Periodic Patterns in Discrete Data Sequences," *Proc. Eighth Pacific-Asia Conf. Knowledge Discovery and Data Mining (PAKDD '04),* 2004.
[16] H. Cao, N. Mamoulis, and D.W. Cheung, "Mining Frequent Spatio-Temporal Sequential Patterns," *Proc. Fifth IEEE Int'l Conf. Data Mining (ICDM '05),* pp. 82-89, 2005.
[17] N. Cressie, *Statistics for Spatial Data.* John Wiley & Sons, 1991.
[18] L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, and J. Vitter, "Scalable Sweeping-Based Spatial Join," *Proc. 24th Int'l Conf. Very Large Databases (VLDB '98),* 1998.
[19] D.J.D.J.M. Patel, "Partition Based Spatial-Merge Join," *Proc. ACM SIGMOD '96,* June 1996.
[20] S.T. Leutenegger and M.A. Lopez, "The Effect of Buffering on the Performance of R-Trees," *Proc. 14th IEEE Int'l Conf. Data Eng. (ICDE '98),* 1998.
[21] N. Koudas and K.C. Sevcik, "Size Separation Spatial Join," *Proc. ACM SIGMOD '97,* 1997.
[22] R. Agarwal and R. Srikant, "Fast Algorithms for Mining Association Rules," *Proc. 20th Int'l Conf. Very Large Data Bases (VLDB '94),* 1994.
[23] "Discovery of Changes from the Global Carbon Cycle and Climate System Using Data Mining," Univ. of Minnesota, http://www.ahpcrc.umn.edu/nasa-umn, 2004.

Yan Huang received the BS degree in computer science from Beijing University, Beijing, in July 1997 and the PhD degree in computer science from the University of Minnesota, Twin Cities, in July 2003. She is currently an assistant professor in the Department of Computer Science and Engineering, University of North Texas, Denton. Her research interests include geosensor networks, spatial databases, and data mining. Her research is supported by Texas Advanced Research Program (ARP), Oak Ridge National Laboratory (ORNL), and the US National Science Foundation (NSF). She is a member of the IEEE, the IEEE Computer Society, the ACM, and the ACM SIGMOD. She is a recipient of the Ralph E. Powe Junior Faculty Enhancement Award from the ORNL Oak Ridge Associated Universities.

Liqin Zhang received the MS degree in computer science from the University of North Texas, Denton, in 2003. She is currently working toward the PhD degree in the Department of Computer Science and Engineering, University of North Texas. Her research interests include spatial databases, data mining, machine learning, and information retrieval.

Pusheng Zhang received the BS degree in computer science from the University of Science and Technology of China, Hefei, China, in 1999 and the PhD degree in computer science from the University of Minnesota, Minneapolis, in 2005. He is currently a software development engineer with the Local Search Team, Microsoft Corporation. His research interests include search engine design, spatial and temporal databases, data mining, and geographic information retrieval. He is a member of the IEEE and the ACM.