RASP: Robust Mining of Frequent Temporal Sequential Patterns under Temporal Variations

Hyunjin Choo
KAIST
Seoul, South Korea
choo@kaist.ac.kr

Minho Eom
KAIST
Daejeon, South Korea
djaalsgh159@kaist.ac.kr

Gyuri Kim
KAIST
Daejeon, South Korea
gyuri2102@kaist.ac.kr

Young-Gyu Yoon
KAIST
Daejeon, South Korea
ygyoon@kaist.ac.kr

Kijung Shin
KAIST
Seoul, South Korea
kijungs@kaist.ac.kr

ABSTRACT

A temporal sequential pattern (TSP) is defined as an ordered collection of events and the corresponding time gaps between consecutive event pairs. When analyzing a sequence of temporal events, identifying frequent TSPs is essential with applications across diverse domains, including neuronal activity analysis, stock trading, and transportation management systems. However, existing mining techniques are often sensitive to hyperparameter settings and may not be scalable for large datasets. Moreover, in practical scenarios, the time gaps in different occurrences of the same TSP may vary to some extent, posing a challenge to the accurate detection of TSPs.

In this work, we propose RASP, a robust and resource-adaptive method for mining frequent TSPs. RASP incorporates (a) duplicated matching between TSPs and instances, based on a novel concept of relaxed TSP, for robustness against variations in time gaps, (b) resource-adaptive automatic hyperparameter tuning for enhancing usability, and (c) a tree-based concise data structure for achieving space efficiency. In our experiments, RASP outperforms four state-of-the-art competitors, offering up to 854× faster speed with similar accuracy and up to 342% greater accuracy at similar speeds.

1 INTRODUCTION

In recent years, there has been growing interest in the analysis of a *temporal event sequence*, which is a sequence of events with occurrence times. Analyzing such sequences, which originate from diverse domains spanning from user behavior in online platforms [16, 44] to neuronal spiking activity in the nervous system [41, 51], allows us to capture the underlying temporal dependencies and dynamics among the events.

In the analysis of temporal event sequences, identifying frequently occurring temporal sequential patterns has been widely employed as a crucial step. A *temporal sequential pattern* (TSP) is defined as an ordered collection of events and the corresponding time gaps between consecutive event pairs, and identifying frequent TSPs has been proven highly valuable in various applications, including:

- **Neuronal Activity Analysis:** Functional groups of neurons associated with specific tasks often exhibit temporally correlated activities [23, 26, 39, 40, 46, 48]. Mining frequent TSPs in a sequence of neuron activation events facilitates the discovery of these groups, revealing how neurons are functionally connected.
- **Stock Trading:** Mining frequent TSPs in a sequence of stock price changes reveals temporal dependencies among multiple stocks [36], aiding investors in improving trading strategies.
- **Transportation Management:** Given a sequence of traffic-congestion events at crossroads [29, 54], mining frequent TSPs helps predict and subsequently alleviate future congestion.

Due to their wide applicability, various methods have been developed for mining frequent TSPs. CAD [40] was developed to detect functionally connected neurons with arbitrary lags and multiple time scales in neuronal recordings. SPADE [39, 46] was designed to detect spatio-temporal spike patterns in spike train data. MIPER [4] was applied to stock and transportation data.

However, these methods share common limitations, including:

- **Lack of Handling Temporal Variatons**: They do not explicitly address temporal variations, which means that the time gaps in a pattern occurrence might not be consistent across various instances. Such variations can stem from various factors, such as measurement errors or inherent system variability, challenging the accurate extraction of frequent TSPs [13].
- **Sensitivity to Hyperparameters**: They are often sensitive to the hyperparameters, such as a predefined threshold [4], resulting in an excessive or insufficient number of TSPs.
- **High Computational and Space Cost**: These methods incur substantial computational and storage costs, especially for handling temporal variations. This is because accurately identifying statistically significant TSPs requires the processing of large-scale datasets that span extended time periods.

In this work, we propose RASP, an algorithm for **R**obust and resource-**A**daptive mining of temporal **S**equential **P**atterns. Specifically, RASP is an approximate algorithm tailored for temporal sequences with temporal variations. RASP is built upon the following ideas, each devised to address the above limitations:

- **Relaxed TSPs and Duplicated Pattern Matching**: For robustness against temporal variations, RASP enables multiple TSPs to share the same instance by using relaxed TSPs, which permit a predefined level of time gap deviation.
- **Resource-Adaptive Automatic Hyperparameter Tuning**: RASP gradually increases the sizes of TSPs to detect larger TSPs. In order to maintain a proper number of TSPs of each size, RASP adaptively adjusts thresholds based on the available resources, enhancing its usability, without requiring a predefined threshold.

- **Tree-based Concise Data Structure**: RASP employs a tree-based compact data structure to efficiently manage the increasing number of TSPs, improving space efficiency.

Our experiments conducted under various scenarios show the effectiveness of RASP, summarized as follows:

- **Accuracy and Robustness**: As depicted in Fig. 1, RASP exhibits a 59% increase in mining accuracy compared to similarly fast competitors under a condition without temporal variations. The margin increases to 342% under temporal variations.
- **Speed**: In the two scenarios in Fig. 1, RASP is 854× and 62× faster, with better accuracy, than the most accurate competitor.
- **Data Requirement**: RASP achieves comparable accuracy with data spanning much shorter time periods than competitors. Collecting data over extended periods is often costly.

Reproducibility. We make the source code and datasets used in the paper available at https://github.com/jin-choo/RASP.

The rest of the paper is organized as follows. In Sect. 2, we review related work. In Sect. 3, we give preliminaries and a problem definition. In Sect. 4, we present our proposed method. In Sect. 5, we review our experiments. In Sect. 6, we make conclusions.

2 RELATED WORK

In this section, we provide a review of related studies.

Co-occurrence Pattern Mining. Frequent pattern mining [1] involves identifying sets of events (e.g., purchases of certain items) that frequently co-occur in datasets, typically based on a minimum support threshold. The Apriori algorithm [3] is a pioneering method that discovers frequent patterns using efficient candidate search based on the anti-monotonicity property. Subsequent works have optimized the mining process for space [10, 37] and speed [42, 47], through hashing [10, 37], partitioning [42], and sampling [47]. Notably, FP-Growth [22] enhances efficiency by eliminating the need for explicit candidate generation, and LCM [49] focuses on patterns with specific properties (e.g., maximality). Refer to surveys [5, 12, 14, 20] for more methods. However, because the order of events is ignored, sequential relationships cannot be captured.

Non-Temporal Sequential Pattern Mining. Sequential pattern mining [2] focuses on finding frequent patterns of events occurring in a specific order (a.k.a., episodes) in a given sequence of events. Algorithms for sequential pattern mining include GSP [45], which extends the A-priori algorithm, and PrefixSpan [21], which employs a pattern-growth approach. SPADE [53] (distinct from SPADE [39, 48] discussed later) further optimizes the mining process by employing vertical database formats, and ClaSP [15] focuses on closed patterns. SPMF [11], an open-source library, offers a wide range of algorithms for sequential (and also co-occurrence) pattern mining. These algorithms primarily focus on the sequence of events, but they do not account for the time intervals between events, which are crucial in many applications.

Temporal Sequential Pattern Mining. While a non-temporal sequential pattern specifies only the order between events, a temporal sequence pattern (TSP) specifies the time gaps between consecutive event pairs within the pattern. The identification of frequent TSPs holds significant utility across diverse applications (refer to Sect. 1 for examples), and as a result, a number of algorithms have emerged for their mining [21, 31, 50, 52].

Russo and Durstewitz [40] developed Cell Assembly Detection (CAD) for detecting functionally connected neurons with arbitrary lags and multiple time scales in neuronal recordings.

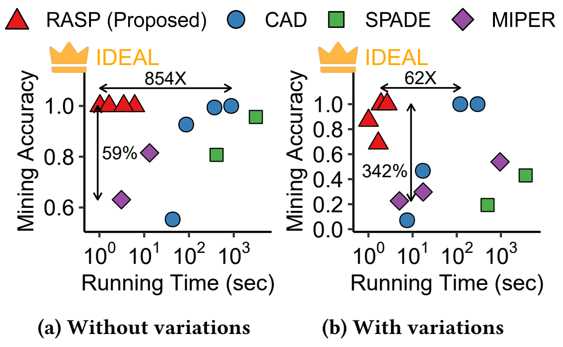


Figure 1: Our proposed RASP gives a significantly better trade-off between accuracy (spec., NDCG@5) and speed than state-of-the-art methods. See Sect. 5.4 for details.

SPADE (Spike PAttern Detection and Evaluation) [39, 48] is designed to detect spatio-temporal spike patterns in spike train data. SPADE employs a frequent itemset mining algorithm to extract spike patterns and assesses their statistical significance based on null distribution estimation. Ao et al. [4] introduced Mining Precise-positioning Episode Rules (MIPER), a tree-based approach used for stock and transportation analysis.

However, while these methods aim to identify frequent TSPs, they commonly suffer from three limitations outlined in Sect. 1, which we aim to overcome. Especially, they do not explicitly consider the temporal variations in the input temporal sequences.

Time Constraints for TSP Mining. Previous works in temporal sequential pattern mining have employed time ranges for the time gaps between consecutive events within a pattern, primarily for constraints. For example, gap-constrained pattern mining [7, 9, 18, 19, 28] extends sequential pattern mining by targeting sequential pattern instances with time gaps that adhere to specific time-range constraints. To identify all patterns satisfying different constraints, however, these methods are required to be performed multiple times with all potential constraints [33].

In the context of TSP mining, the following ideas have been used: (a) constraining the time span of each TSP instance to a predefined time range [17, 24, 25, 35], (b) constraining each time gap to fall within a predefined set of disjoint time ranges [30], for example by segmenting timestamps into disjoint ranges [40, 48].

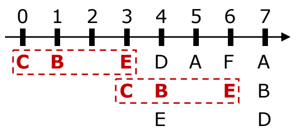
Our proposed method, RASP, also incorporates time ranges for time gaps, but with duplicated matching between TSPs and instances based on a novel concept of relaxed TSP. This novel strategy enables an instance to be matched with multiple TSPs, enhancing robustness against temporal variations.

3 PRELIMINARY AND PROBLEM DEFINITION

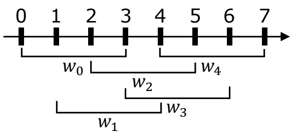
In this section, we introduce some basic concepts and notations frequently used throughout the paper, which are also summarized in Table 1. Then, we present the problem that we aim to address.

3.1 Preliminary Concepts

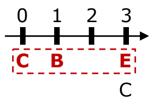
Temporal Event Sequences. Let $\mathcal{E}$ be a set of *events*, and let (E, t) be an instance of an event $E \in \mathcal{E}$ (e.g., a spike of a specific neuron) that occurs at the time $t \in \mathbb{N}_0$. A *temporal event sequence* is defined as a sequence of event instances ordered by their occurrence time, denoted as $\bar{s} = \langle(\bar{E}_1, \bar{t}_1), (\bar{E}_2, \bar{t}_2), \cdots, (\bar{E}_n, \bar{t}_n)\rangle$, where $\bar{E}_i \in \mathcal{E}$, $1 \le \forall i \le n$, and $\bar{t}_1 \le \bar{t}_2 \le \cdots \le \bar{t}_n$.



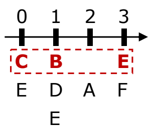
(a) A temporal event sequence and a TSP



(b) Windows of time span $L = 4$



(c) Window w_0



(d) Window w_3

Figure 2: Toy examples of a temporal event sequence, a temporal sequential pattern (TSP), and windows.

> **Example 1.** Fig. 2(a) shows a temporal event sequence $\bar{s} = \langle (C, 0), (B, 1), (C, 3), (E, 3), (B, 4), (D, 4), (E, 4), (A, 5), (E, 6), (F, 6), (A, 7), (B, 7), (D, 7)\rangle$ for an event set $\mathcal{E} = \{A, B, C, D, E, F\}$.

Temporal Sequential Patterns. A *temporal sequential pattern* (TSP) $\alpha = (\langle E_1, E_2, \cdots, E_l\rangle, \langle \Delta t_1, \Delta t_2, \cdots, \Delta t_{l-1}\rangle)$ of size l is defined as a pair of ordered sets of (a) distinct events[1] and (b) non-negative *time gaps* between two consecutive events. That is, $E_i \in \mathcal{E}, 1 \le \forall i \le l$, $E_i \ne E_j$, $1 \le \forall i \ne j \le l$, and $\Delta t_i \in \mathbb{N}_0, 1 \le \forall i \le l-1$.

Occurrences, Instances, and Support. A subsequence $s' = \langle (E'_1, t'_1), (E'_2, t'_2), \cdots, (E'_n, t'_n)\rangle$ of $\bar{s}$ is an *occurrence* (or *instance*) of α if $E'_i = E_i, 1 \le \forall i \le l$, and $t'_{i+1} - t'_i = \Delta t_i, 1 \le \forall i \le l-1$, implying:

$$t'_{i+1} - t'_1 = \sum_{j=1}^{i} \Delta t_j, 1 \le \forall i \le l-1. \tag{1}$$

The number of occurrences of α is referred to as the *support* of α and denoted as $sup(\alpha)$.

> **Example 2.** Consider a TSP $\alpha = (\langle C, B, E\rangle, \langle 1, 2\rangle)$, which is highlighted in Fig. 2(a). The occurrences of α are $\langle (C, 0), (B, 1), (E, 3)\rangle$ and $\langle (C, 3), (B, 4), (E, 6)\rangle$, and the support of α is $sup(\alpha) = 2$.

Pattern Frequency Measures. The frequency, or significance, of a pattern can be assessed by comparing its measured support with the expected support under the assumption of independent event occurrences. The expected support of a pattern α, denoted as $sup_{exp}(\alpha)$, over a given temporal event sequence $\bar{s}$, represents the anticipated value of the support of α. It can be calculated by considering the frequency of each event $E_i, 1 \le \forall i \le l$ in α and the time span $T = \bar{t}_n - \bar{t}_1 + 1$ of $\bar{s}$:

$$sup_{exp}(\alpha) = T \times \prod_{i=1}^{l} \frac{sup(E_i)}{T} = \frac{\prod_{i=1}^{l} sup(E_i)}{T^{l-1}}. \tag{2}$$

In addition to support, we introduce two more frequency measures for a pattern: (a) *leverage* [38]: the difference between a pattern's measured and expected support, and (b) *lift* [6]: the ratio of a pattern's measured support to its expected support. The *leverage* of α is defined as $lev(\alpha) = sup(\alpha) - sup_{exp}(\alpha)$;and the *lift* of α is defined as $lift(\alpha) = sup(\alpha)/sup_{exp}(\alpha)$.

[1]While we focus on TSPs with distinct events for clarity of presentation, our concepts and algorithms can be easily extended to handle duplicates, as discussed in Section B of the online appendix [8].

Table 1: Frequently-used notations and their definitions.

$\bar{s}$	Temporal event sequence with temporal variations
l	Target size of a TSP
k	Target number of TSPs
δ	Maximum time gap between two consecutive events
L	Maximum time span of a TSP
I	Tolerance against temporal variations
W	Mapping between pairs of (event, relative occurrence time) and their WIDs
F_1	Mapping between events and their WIDs
m_i	Maximum number of TSPs of size i
F_i	Mapping between frequent TSPs of size i and (WIDs, frequency) pairs

We use the notation $freq$ to represent a frequency measure, encompassing support, leverage, or lift. The frequency value of each TSP α in sequence $\bar{s}$ is denoted as $freq_\alpha$.

> **Example 3.** For the TSP $\alpha = (\langle C, B, E\rangle, \langle 1, 2\rangle)$ in Fig. 2(a), $sup_{exp}(\alpha) = 8 \cdot \frac{2}{8} \cdot \frac{3}{8} \cdot \frac{3}{8} \approx 0.28$, $lev(\alpha) \approx 1.72$, and $lift(\alpha) \approx 7.11$.

3.2 Problem Definition

Below, we define the concept of temporal variations and the problem of robust TSP mining.

Temporal Variations. We consider a ground-truth sequence $\tilde{s} = \langle (\tilde{E}_1, \tilde{t}_1), (\tilde{E}_2, \tilde{t}_2), \cdots, (\tilde{E}_n, \tilde{t}_n)\rangle$ without temporal variations. Temporal variations in $\tilde{s}$ result in the input temporal event sequence $\bar{s} = \langle (\bar{E}_1, \bar{t}_1), (\bar{E}_2, \bar{t}_2), \cdots, (\bar{E}_n, \bar{t}_n)\rangle$ that is close but not identical to $\tilde{s}$, i.e., there exists a one-to-one mapping $f : \{1, \cdots, n\} \to \{1, \cdots, n\}$ where $\bar{E}_{f(i)} = \tilde{E}_i$ and $\bar{t}_{f(i)} \approx \tilde{t}_i, 1 \le \forall i \le n$. Without temporal variations, $\bar{s} = \tilde{s}$ holds.

Problem Definition: Robust Mining of TSPs.

> - **Given:**
>   (1) a temporal event sequence $\bar{s}$ with temporal variations,
>   (2) the target size l of each TSP,
>   (3) the target number k of TSPs,
> - **Find:** k-most frequent TSPs of size l on the ground-truth temporal event sequence $\tilde{s}$ ($\tilde{s}$ is unobservable and without temporal variations),
> - **Constraints:**
>   (1) max time gap δ between consecutive events,
>   (2) max time span L of a TSP.

In our problem definition, we assume that the temporal events in a TSP are not too widely distributed over time, and hence we impose two constraints on TSPs: the *max time gap* δ between any two consecutive events within a TSP (i.e., $\Delta t_i = t_{i+1} - t_i \le \delta, 1 \le \forall i \le l-1$) and the *max time span* L of a TSP (i.e., $\sum_{i=1}^{l-1} \Delta t_i = t_l - t_1 \le L$). These constraints are used to target TSPs within a reasonable time span, which are of interest in many applications. Moreover, these constraints can be specified based on the characteristics of datasets.

Because the ground-truth temporal event sequence $\tilde{s}$ without temporal variations is unobservable, achieving precise and exhaustive identification of TSPs is exceptionally challenging. Given this difficulty, previous works have concentrated on designing approximate methods [4, 40, 48], a pursuit we also undertake.

4 PROPOSED METHOD

In this section, we present our proposed method, RASP (**R**obust and resource-**A**daptive mining of temporal **S**equential **P**atterns).

```
Algorithm 1: Overview of RASP
  Input: (1) $\bar{s}$: a temporal event sequence,
    (2) $l$: the target size of each TSP,
    (3) $k$: the target number of TSPs,
    (4) $L$: max time span of a TSP,
    (5) $\delta$: max time gap between consecutive events,
    (6) $I$: tolerance against temporal variations
  Output: top-$k$ frequent TSPs of size $l$
1 preprocess $\bar{s}$ and compute $W$ and $F_1$    // Alg. 2
    // $W$: mapping between (event, relative occurrence
    time) pairs and their WIDs, and $F_1$: mapping between
    events and their WIDs
2 for $i \leftarrow 2$ to $l - 1$ do
3     $m_i \leftarrow$ max number of TSPs of size $i$    // Eq. (4)
4     $F_i \leftarrow m_i$ most frequent TSPs of size $i$ from $W$ & $F_{i-1}$
          // Alg. 3 and 4, $F_i$: mapping between frequent
      TSPs of size $i$ and (WIDs, frequency) pairs
5     remove $F_{i-1}$
6 $F_l \leftarrow k$ most frequent TSPs of size $l$ from $W$ & $F_{l-1}$
7 return $F_l$
```

```
Algorithm 2: Preprocessing
  Input: (1) $\bar{s}$: a temporal event sequence,
    (2) $L$: max time span of a TSP
  Output: (1) $W$: mapping between (event, relative occurrence
        time) pairs and their WIDs,
    (2) $F_1$: mapping between events and their WIDs
1 $W \leftarrow \emptyset, F_1 \leftarrow \emptyset, T \leftarrow$ time span of $\bar{s}$
2 for $t \leftarrow 0, \cdots, T - L$ do
3     foreach $(E, t) \in \bar{s}$ do
4         if $E \in F_1.keys()$ then
5             $F_1[E] \leftarrow F_1[E] \cup \{t\}$
6         else
7             $F_1[E] \leftarrow \{t\}$
8         $w_t \leftarrow \{(E', t') : (E', t + t') \in \bar{s}, 0 \le t' < L\}$
9         foreach $(E', t') \in w_t$ do
10            if $(E', t') \in W.keys()$ then
11                $W[(E', t')] \leftarrow W[(E', t')] \cup \{t\}$
12            else
13                $W[(E', t')] \leftarrow \{t\}$
14    return $W, F_1$
```

4.1 Overview (Alg. 1)

We provide an overview of RASP in Alg. 1. Given a temporal event sequence $\bar{s}$, RASP returns the k most frequent TSPs (in terms of a frequency measure $freq$) of size l. Note that the values of k and l are defined in the problem statement (refer to Sect. 3.2); these are not hyperparameters of the proposed method. RASP first preprocesses $\bar{s}$ to obtain W (defined in Sect. 4.2). Then, it identifies frequent TSPs of each size by increasing the size from 1 to l one by one. RASP determines the maximum number m_i of TSPs of each size i in a resource-adaptive manner (described in Sect. 4.4). Below, we first present the preprocessing step and then describe three key ideas behind RASP: (a) relaxed TSPs and duplicated pattern matching, (b) resource-adaptive automatic hyperparameter tuning, and (c) a tree-based concise data structure, as briefly introduced in Sect. 1.

4.2 Preprocessing (Alg. 2)

We outline the preprocessing step of RASP in Alg. 2, where RASP transforms the input temporal event sequence $\bar{s}$ into mappings W and F_1. These mappings facilitate the efficient growth of TSPs, as discussed in the following subsections.

Concepts: Windows and WIDs. Imagine a 'window' of time span L is sliding through the input temporal event sequence (see Fig. 2(b) for an example). Within this window, each event instance is represented as a pair (E, t'), where $E \in \mathcal{E}$ denotes an event and $0 \le t' < L$ indicates its relative occurrence time within the window. The set of pairs indicating the event instances within the window starting at time t is denoted as $w_t = \{(E, t') : (E, t + t') \in \bar{s}, E \in \mathcal{E}, 0 \le t' < L\}$. We refer to the starting time of a window as the *window ID* (WID).

Example 4. In Fig. 2, $w_0 = \{(C, 0), (B, 1), (C, 3), (E, 3)\}$ and $w_3 = \{(C, 0), (E, 0), (B, 1), (D, 1), (E, 1), (A, 2), (E, 3), (F, 3)\}$. Note that the sequence $\langle(C, 0), (B, 1), (E, 3)\rangle$, which is highlighted in Figs. 2(a), (c)-(d), corresponds to the TSP $\alpha = (\langle C, B, E\rangle, \langle 1, 2\rangle)$.

Computing W (Lines 8-13 of Alg. 2). In lines 8-13, RASP constructs the mapping W between each (event, relative occurrence time) pair and the WIDs of the windows that contain the pair. Note that only the windows where at least one event occurs at its beginning are considered. Our preprocessing scheme, technically, extends [3], designed for non-sequential pattern mining.

Example 5. In Fig. 2, $W[(C, 0)] = \{0, 3\}$, $W[(B, 1)] = \{0, 3, 6\}$, and $W[(E, 3)] = \{0, 1, 3\}$. Note that $W[(C, 0)] \cap W[(B, 1)] \cap W[(E, 3)] = \{0, 3\}$, which are the WIDs of the windows where the TSP $\alpha = (\langle C, B, E\rangle, \langle 1, 2\rangle)$ starts occurring at their beginning (i.e., w_0 and w_3).

Computing F_1 (Lines 3-7 of Alg. 2). In lines 3-7, we construct the mapping F_1, which is the mapping between each event and the WIDs of the windows where the event occurs at their beginning.

4.3 Relaxed TSPs and Duplicated Pattern Matching (Algs. 3 and 4)

Key Idea. To improve robustness against temporal variations, RASP allows for a predefined level of time gap deviations and also permits an instance to be matched with multiple TSPs.

Specifically, to address temporal variations in the time gaps in a TSP (defined in Sect. 3.1), we introduce the concept of a *relaxed TSP*. Based on this concept, RASP matches each instance with potentially multiple TSPs and eventually grows the size of frequent TSPs, as described in Algs. 3 and 4.

Concepts: Relaxed TSPs. Let $\Delta\mathcal{T} = \{\Delta\bar{T}_0, \Delta\bar{T}_1, \cdots\}$ be the set of all possible relaxed time gaps. A *relaxed TSP* $\alpha = (\langle E_1, E_2, \cdots, E_l\rangle, \langle \Delta T_1, \Delta T_2, \cdots, \Delta T_{l-1}\rangle)$ of size l is defined as a pair of ordered sets of (a) distinct events and (b) relaxed time intervals. That is, $E_i \in \mathcal{E}, 1 \le \forall i \le l$, $E_i \ne E_j$, $1 \le \forall i \ne j \le l$, and $\Delta T_i \in \Delta\mathcal{T}, 1 \le \forall i \le l - 1$. A subsequence $s' = \langle(E'_1, t'_1), (E'_2, t'_2), \cdots, (E'_n, t'_n)\rangle$ of $\bar{s}$ is an *occurrence* (or *instance*) of α if $E'_i = E_i, 1 \le \forall i \le l$, and

$$\sum_{j=1}^{i} \Delta T_j - I \le t'_{i+1} - t'_1 < \sum_{j=1}^{i} \Delta T_j + I, 1 \le \forall i \le l - 1, \quad (3)$$

where I is a constant for tolerance against temporal variations. Note that the conditions in Eq. (3) relax that in Eq. (1), which is used for TSPs. Throughout the paper, we set $\Delta\bar{T}_i = i \times I, \forall i \in \mathbb{N}_{\not{o}}$. The notions of support and other frequency measures are trivially extended to relaxed TSPs. Note that the same subsequence can be an occurrence of multiple relaxed TSPs.

Algorithm 3: Mining Frequent TSP (Size 2)

```
   Input: (1) $W$: mapping between (event, relative occurrence time)
              pairs and their WIDs,
          (2) $F_1$: mapping between events and their WIDs,
          (3) $m_2$: max number of TSPs of size 2,
          (4) $\delta$: max time gap between consecutive events,
          (5) $I$: tolerance against temporal variations
   Output: $F_2$: mapping between frequent TSPs of size 2 and
              (WIDs, frequency) pairs
 1 $n \leftarrow 0; \theta \leftarrow 2$        // $n$: current count, $\theta$: threshold
 2 $F_2 \leftarrow \emptyset$
 3 foreach $E_1 \in F_1.keys()$ do
 4     foreach $E_2 \in \mathcal{E} \setminus \{E_1\}$ do
 5         for $t \leftarrow 0$ to $\lfloor \delta/I \rfloor$ do
 6             $\alpha \leftarrow (\langle E_1, E_2 \rangle, \langle tI \rangle)$   // $tI = \Delta\bar{T}_t$ (Sect. 4.3)
 7             $W_\alpha \leftarrow F_1[E_1] \cap (\cup_{w=-I}^{I-1} W[(E_2, \min(tI + w, \delta))])$
 8             compute $freq_\alpha$ from $W_\alpha$
 9             if $freq_\alpha \geq \theta$ then
10                 $F_2[\alpha] \leftarrow (W_\alpha, freq_\alpha)$
11                 increment $n$
12                 if $n > m_2$ then                    // Sect. 4.4
13                     increment $\theta$ by a predefined gap
14                     prune TSPs in $F_2$ whose frequency $< \theta$
15                     $n \leftarrow n -$number of pruned TSPs
16     return $F_2$
```

Example 6. In Fig. 2, if $I = 2$, a subsequence $\langle (C, 0), (B, 1), (E, 3) \rangle$ is an occurrence of the following relaxed TSPs: $(\langle C, B, E \rangle, \langle 0, 2 \rangle)$, $(\langle C, B, E \rangle, \langle 2, 2 \rangle)$, $(\langle C, B, E \rangle, \langle 0, 4 \rangle)$, and $(\langle C, B, E \rangle, \langle 2, 4 \rangle)$.

Mining Frequent TSPs of Size 2 (Alg. 3). Alg. 3 describes how RASP discovers frequent TSPs of size 2. In the first part (lines 4-7), RASP attempts to expand each TSP of size 1 (i.e., an event instance) in F_1 by appending an event instance that satisfies Eq. (3) and the constraints specified in our problem definition. As shown in line 4, the search for such an event instance is performed using only W and F_1. From them, the WIDs of the windows at the beginning of which the resulting TSP α of size 2 starts occurring (denoted by W_α) can be obtained through several set operations.

Example 7. In Fig. 2, if $I = 2$, for a relaxed TSP $\alpha = (\langle C, B \rangle, \langle 0 \rangle)$, $W_\alpha = F_1[C] \cap (W[(B, 0)] \cup W[(B, 1)]) = \{0, 3\}$. For $\alpha = (\langle C, B \rangle, \langle 2 \rangle)$, $W_\alpha = F_1[C] \cap (W[(B, 0)] \cup W[(B, 1)] \cup W[(B, 2)] \cup W[(B, 3)]) = \{0, 3\}$.

In the remaining part (lines 8-15), RASP retains at most m_2 most frequent TSPs of size 2. To accomplish this, it incrementally raises the threshold θ. The process for determining the budget m_2 is presented in Sect. 4.4. As the output, the mapping F_2 between frequent TSPs of size 2 and (WIDs, frequency) pairs is returned.

Mining Frequent TSPs of Size i (> 2) (Alg. 4). Alg. 4 depicts how RASP discovers frequent TSPs of a larger size. It follows the same structure as Algorithm 3: (a) growing a TSP of size $i - 1$ to i while satisfying both Eq. (3) and the constraints (lines 3-12) and (b) retaining at most m_i most frequent TSPs of size i by gradually raising the threshold (lines 13-20). Note that F_i is computed from F_{i-1} and W without accessing input temporal event sequence $\bar{s}$.

Algorithm 4: Mining Frequent TSP (size i)

```
   Input: (1) $W$: mapping between (event, relative occurrence time)
              pairs and their WIDs,
          (2) $F_{i-1}$: mapping between frequent TSPs of size $(i-1)$ and
          (WIDs, frequency) pairs,
          (3) $m_i$: max number of TSPs of size $i$,
          (4) $\delta$: max time gap between consecutive events,
          (5) $L$: max time span of a TSP,
          (6) $I$: tolerance against temporal variations
   Output: $F_i$: mapping between frequent TSPs of size $i$ and (WIDs,
              frequency) pairs
 1 $n \leftarrow 0; \theta \leftarrow 2$        // $n$: current count, $\theta$: threshold
 2 $F_i \leftarrow \emptyset$
 3 foreach $\alpha' \in F_{i-1}.keys()$ do
 4     $(\langle E_1, \cdots, E_{i-1} \rangle, \langle \Delta t_1, \cdots, \Delta t_{i-2} \rangle) \leftarrow \alpha'$
 5     $(W_{\alpha'}, freq_{\alpha'}) \leftarrow F_{i-1}[\alpha']$
 6     foreach $E_i \in \mathcal{E} \setminus \{E_1, \cdots, E_{i-1}\}$ do
 7         for $t \leftarrow 0$ to $(\lfloor \min(\delta, L - \sum_{j=1}^{i-2} \Delta t_j)/I \rfloor)$ do
 8             $\Delta t_{i-1} = tI; t_i = \sum_{j=1}^{i-1} \Delta t_j$
 9             $\alpha'' \leftarrow (\langle E_2, \cdots, E_i \rangle, \langle \Delta t_2, \cdots, \Delta t_{i-1} \rangle)$
10             if $\alpha'' \in F_{i-1}.keys()$ then
11                 $\alpha \leftarrow (\langle E_1, \cdots, E_i \rangle, \langle \Delta t_1, \cdots, \Delta t_{i-1} \rangle)$
12                 $W_\alpha \leftarrow$
                    $W_{\alpha'} \cap (\cup_{w=-I}^{I-1} W[(E_i, \min(t_i + w, \delta))])$
13                 compute $freq_\alpha$ from $W_\alpha$
14                 if $freq_\alpha \geq \theta$ then
15                     $F_i[\alpha] \leftarrow (W_\alpha, freq_\alpha)$
16                     increment $n$
17                     if $n > m_i$ then                // Sect. 4.4
18                         increment $\theta$ by a predefined gap
19                         prune TSPs in $F_i$ whose frequency $< \theta$
20                         $n \leftarrow n -$number of pruned TSPs
21     return $F_i$
```

4.4 Resource-Adaptive Automatic Hyperparameter Tuning

Key Idea. To maintain a proper number of frequent TSPs, RASP automatically and adaptively adjusts frequency thresholds based on the available resources, improving its usability.

To determine the number of frequent TSPs of each size to retain, existing methods rely on hyperparameters, such as using a predefined threshold [4], a predetermined condition [48], or conducting statistical testing with a predefined significance level [40]. They are often sensitive to hyperparameters, resulting in an excessive number of TSPs, which causes out-of-memory problems, or an insufficient number of TSPs. Finding suitable hyperparameter values, which vary across datasets, necessitates extensive trial and error.

To retain a sufficient number of TSPs but without exceeding storage capacity, RASP adaptively adjusts the threshold θ, as described in the previous subsection (lines 8-15 of Alg. 3 and lines 13-20 of Alg. 4) and illustrated below. This eliminates the need for extensive trial and error, enabling RASP to be effective for various datasets.

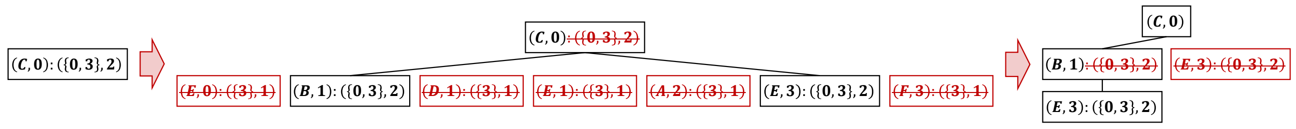
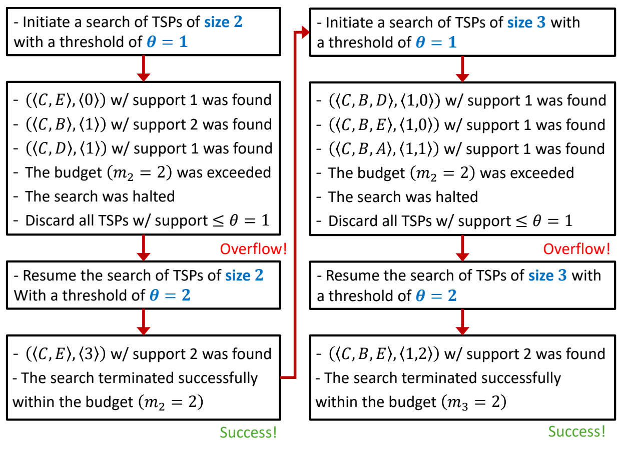


Figure 3: Tree representations of TSPs that start with an event C for the example in Fig. 2, when the threshold θ is 2. Pruned elements are marked in red strikethrough. The figure displays the trees after computing F_1, F_2, and F_3 (from left to right).


- Initiate a search of TSPs of size 2 with a threshold of $\theta = 1$

- $(\langle C, E\rangle, \langle 0\rangle)$ w/ support 1 was found
- $(\langle C, B\rangle, \langle 1\rangle)$ w/ support 2 was found
- $(\langle C, D\rangle, \langle 1\rangle)$ w/ support 1 was found
- The budget ($m_2 = 2$) was exceeded
- The search was halted
- Discard all TSPs w/ support $\leq \theta = 1$

Overflow!

- Resume the search of TSPs of size 2 With a threshold of $\theta = 2$

- $(\langle C, E\rangle, \langle 3\rangle)$ w/ support 2 was found
- The search terminated successfully within the budget ($m_2 = 2$)

Success!

- Initiate a search of TSPs of size 3 with a threshold of $\theta = 1$

- $(\langle C, B, D\rangle, \langle 1,0\rangle)$ w/ support 1 was found
- $(\langle C, B, E\rangle, \langle 1,0\rangle)$ w/ support 1 was found
- $(\langle C, B, A\rangle, \langle 1,1\rangle)$ w/ support 1 was found
- The budget ($m_2 = 2$) was exceeded
- The search was halted
- Discard all TSPs w/ support $\leq \theta = 1$

Overflow!

- Resume the search of TSPs of size 3 with a threshold of $\theta = 2$

- $(\langle C, B, E\rangle, \langle 1,2\rangle)$ w/ support 2 was found
- The search terminated successfully within the budget ($m_3 = 2$)

Success!

Figure 4: A toy example of resource-adaptive automatic hyperparameter tuning by RASP. See Example 8 for details.

Example 8. In Fig. 4, we illustrate the resource-adaptive automatic hyperparameter tuning process for the example in Fig. 2, under the condition that (a) the total number M of TSPs that can be retained is set to 4, and (b) the maximum number m_i of TSPs allowed for each size i is set to 2. During the mining process for TSPs of sizes 2 and 3, the frequency threshold θ is initially set to 1 but then adaptively adjusted to ensure that no more than 2 TSPs are retained for each size.

Specifically, for each size $i \in \{2, \cdots, l\}$, RASP computes the maximum number m_i of TSPs of size i to retain (line 3 of Alg. 1), based on the available resources, as follows:

$$m_i = \begin{cases} (M - |W|)/2, & 2 \leq i < l - 1 \\ M - |W| - \max(|F_{l-2}|, k), & i = l - 1 \\ k, & i = l, \end{cases} \tag{4}$$

where M denotes the total number of TSPs that can be retained. The rationales behind Eq. (4) are as follows:

(1) For each $2 \leq i < l - 1$, only W, F_{i-1}, and F_i need to be retained, and thus we set m_i's so that their cardinality sum (which is roughly proportional to storage requirements) stays within the budget M, i.e., ensuring that the following equation holds:

$$|W| + |F_{i-1}| + |F_i| \leq |W| + m_i + m_{i-1} = M. \tag{5}$$

(2) For each $2 \leq i < l - 1$, we balance the budgets for F_{i-1} and F_i, i.e., $m_{i-1} = m_i$.
(3) For $i \geq l - 1$, we adjust the formula for m_i based on the fact that we only need to maintain k TSPs of size l according to the problem definition. Note that Eq. (5) still holds.

4.5 Tree-based Concise Data Structure

Key Idea. To efficiently manage the growing number of TSPs, RASP utilizes a tree-based compact data structure.

RASP stores frequent TSPs (i.e., F_i) in the form of a set of trees (i.e., forest) so that TSPs share a common prefix based on the order of event instances, thereby reducing space usage. Specifically, each tree encodes frequent TSPs as follows: (a) each node represents a pair of an event and its relative occurrence time, (b) each leaf node corresponds to the TSP consisting of events and time gaps arranged in sequence along the path from the root to the leaf node, and (c) each leaf node additionally contains the set of WIDs of the windows at the beginning of which the corresponding TSP starts occurring and the frequency.

During the execution of RASP, the trees are retained in a compact form without unnecessary nodes. Specifically, while computing F_i, leaf nodes with depth i that do not correspond to the top-m_i TSPs are pruned. After computing F_i, leaf nodes with depth less than i are also pruned, as they are no longer utilized.

Example 9. In Fig. 3, we illustrate the transformation of a tree, rooted at $(C, 0)$ for the example shown in Fig. 2, under the condition that the threshold θ is set to 2. Consider the tree displayed in the middle of Figure 3. By disregarding the pruned elements marked in red strikethrough, this tree encodes two TSPs: $(\langle C, B\rangle, \langle 1\rangle)$ and $(\langle C, E\rangle, \langle 3\rangle)$. Both TSPs have the same corresponding WIDs $\{0, 3\}$, and their frequency (specifically, support) is 2.

4.6 Complexity Analysis

Below, we analyze the time and space complexity of RASP.

The computational bottleneck of Alg. 4 is line 12, which contains $O(I)$ union operations, each of which takes $O(T)$ time, where T is the time span of the input temporal event sequence $\bar{s}$. Thus, the time complexity of the line is $O(I \cdot T)$, and since $|F_{i-1}| \leq m_{i-1} = O(M)$, it is executed $O(M \cdot |\mathcal{E}| \cdot \delta / I)$ times. Hence, the time complexity of Alg. 4 is $O(M \cdot |\mathcal{E}| \cdot \delta \cdot T)$. RASP (i.e., Alg. 1) executes Alg. 4 $O(l)$ times, where l is the target TSP size, and this dominates the other parts. Thus, the time complexity of RASP is $O(M \cdot |\mathcal{E}| \cdot \delta \cdot T \cdot l)$.

The space required by RASP is determined by the size of the preprocessed results and the number of frequent TSPs retained by RASP. As detailed in the previous subsections, the number of the mappings (i.e., the preprocessed results) is $|W|$, and the number of retained frequent TSPs during the mining of TSPs of size i is $|F_{i-1}| + |F_i| \leq m_{i-1} + m_i$. Eq. (4) ensures that $|W| + m_{i-1} + m_i \leq M$. The primary memory requirement per mapping and TSP is the bit vector of size $O(T)$ representing the WIDs of the windows at the beginning of which the corresponding TSP begins. Therefore, the space requirement of RASP is $O(M \cdot T)$.

5 EXPERIMENTS

In this section, we present the results of our experiments designed to answer the following questions:

- **Q1. Accuracy**: How accurately does RASP discover frequent TSPs, especially under temporal variations?
- **Q2. Scalability**: How does RASP scale depending on the time span and event count of the input data?
- **Q3. Speed-Accuracy Trade-off**: Does RASP give a better speed-accuracy trade-off than its competitors?
- **Q4. Ablation Study**: Does duplicated pattern matching contribute to the accuracy of RASP?

5.1 Experimental Settings

Machines. We used a server with 3.8 GHz AMD Ryzen 9 3900X CPUs and 128 GB RAM.

Datasets. For main experiments, we used neuron activity datasets generated by the CN2 simulator[2]. In these datasets, each event indicates the spike of a specific neuron. The simulator generates ground-truth frequent TSPs together with non-TSP event instances (i.e., independent spikes of neurons), which pose a challenge in identifying the frequent TSPs. Specifically, the CN2 simulator creates each ground-truth TSP with randomly-chosen event compositions and time gaps between consecutive events. Different ground-truth TSPs have distinct event compositions and time gaps. The instances of each ground-truth TSP recur throughout datasets, as specified in Table 2. With the simulator, we can introduce temporal variations by adding zero-mean Gaussian noise with a predefined standard deviation to each time gap in the ground-truth TSP instances. Moreover, we can introduce probabilistic participation of events into TSPs, mirroring real-world situations [34], by letting each event composing a TSP appear in an instance with a predefined probability. Specifically, we considered the following dataset settings.

- **S1. Variation-Free**: Settings in Table 2 without temporal variations or probabilistic participation.
- **S2. Variations**: S1 with temporal variations with a specified standard deviation.
- **S3. Event Count**: S1 with a specified number of events. In general, as the number of events increases, finding TSPs becomes more challenging.
- **S4. Mixed-Easy**: S1 with temporal variations with a standard deviation of 10 ms and probabilistic participation with a probability of 0.8. The occurrence rate of TSPs is raised to 0.4 Hz for increased significance.
- **S5. Mixed-Hard**: S1 with temporal variations with a standard deviation of 30 ms and probabilistic participation with a probability of $2/3$. The occurrence rate of TSPs is 0.2 Hz.

For each setting, we created three datasets with different random seeds, all the results were averaged across them.

Method Setups. In addition to RASP, as the competing methods, we tested CAD [40], SPADE [48], and MIPER [4], described in Sect. 2. For all methods, we set the maximum time gap δ to 0.1 secs, which corresponds to that of each GT TSP in the datasets, and the maximum time span of a TSP L to (the size of each GT TSP $-1) \times \delta = 0.4$ secs. Since the timestamps in the datasets are continuous and thus unsuitable for the considered methods, we discretized them into bins with various intervals and reported the best result. We commonly applied intervals of 15, 25, and 40 ms. For CAD, we additionally included intervals of 60 and 85 ms, as CAD benefits from a broader range of time bins,[3] while the others do not (refer to Appendix A.3 for the empirical effects of bin sizes).

[2]https://github.com/NICALab/CN2-Simulator

Table 2: Default setting of neuron activity datasets.

Total number of events (i.e., neurons) ($\lvert\mathcal{E}\rvert$)	50
Time span of data (T)	10, 50, 100, 500, 1000 secs
Occurrence rate of non-TSP event	2.0 Hz
Number of ground-truth (GT) TSPs	5
Occurrence rate of GT TSPs	0.2 Hz
Size of each GT TSP	5*
Time gap between events in a GT TSP	$\sim U(0, 0.1)$ secs

* Larger TSPs are considered in Appendix A.2.

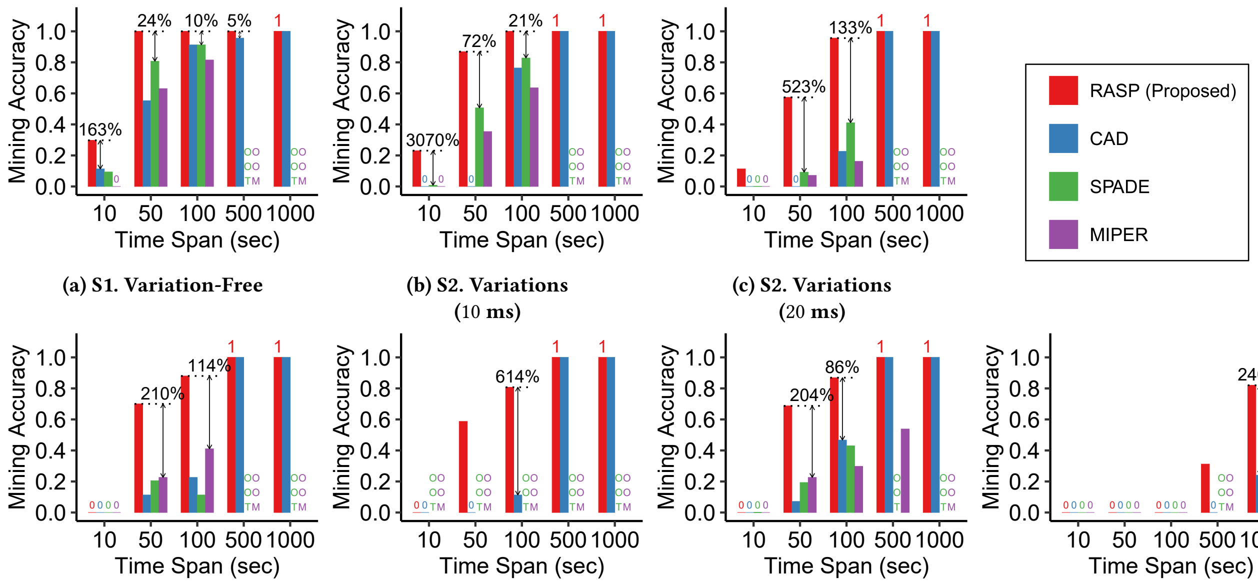
In RASP, we used a bit array to store WIDs for each TSP. Each bit array requires memory space proportional to the number of all WIDs in the datasets. Therefore, we set the maximum number M of TSPs to retain (see Sect. 4.4) to a constant (spec., 10^9) divided by the total number of WIDs. Refer to Appendix A.1 for the effect of values of M. We set the tolerance I against temporal variations to 1 time bin for Variation-Free (S1) and 2 time bins for the other settings. We set k, the target number of frequent TSPs, to $1,000$.

For resource-adaptive thresholding, where frequency measures are extensively used, we employed support as the frequency measure $freq$, which is computationally cheap. However, since support values are integers, there can be many tied TSPs with the same support. Thus, for the final top-k selection, we used leverage (see Sect. 3), instead. We did not use lift in our experiments, because leverage led to better performance. However, lift or other frequency measures can be more suitable depending on applications, so RASP remains flexible regarding the choice of frequency measures.

Method Setup Details of Competitors. To enhance accuracy, we created four additional versions of CAD based on the modified assumptions, statistical corrections, or statistical test methods for our experiments. Various significance levels ($\alpha = 0.01, 0.05, 0.1, 0.2,$ and 1.0) were applied to all these versions of CAD, and we reported the best value obtained across all trials using all versions. For SPADE, the number of surrogates was set to 2×(the number of events), and the dithering values were set to 20, 50, and 100 ms. We also reported the best value obtained across all trials. Furthermore, we modified MIPER to be more suitable for our experiments by changing the target output from episode rules to TSPs and the significance measure from confidence to leverage. We identified the top-k TSPs based on their significance measures (leverage for MIPER and p-value for CAD and SPADE). Note that our proposed method, RASP and MIPER[4] were implemented in Java, CAD[5] was implemented in MATLAB, and SPADE[6] was implemented in C++ and Python.

Evaluation Metrics. We assessed the mining accuracy of all methods using *normalized discounted cumulative gain* (NDCG) and *recall* (RC). NDCG@n measures the quality of the top-n ranking by comparison with ground-truth TSPs, while RC@n quantifies the fraction of ground-truth frequent TSPs identified by RASP among all ground-truth frequent TSPs. Both metrics ranges from 0.0 to 1.0, with higher values indicating better ranking quality. After examining NDCG@n and recall@n values for $n =$

[3]Instead of using a single fixed time bin, CAD iterates through each time bin in a user-defined range and selects the optimal value.
[4]https://github.com/aoxaustin/MIPER/
[5]https://github.com/DurstewitzLab/Cell-Assembly-Detection
[6]https://viziphant.readthedocs.io/en/latest/



(a) S1. Variation-Free

(b) S2. Variations (10 ms)

(c) S2. Variations (20 ms)

(d) S3. Event Count (100)

(e) S3. Event Count (1, 000)

(f) S4. Mixed-Easy

(g) S5. Mixed-Hard

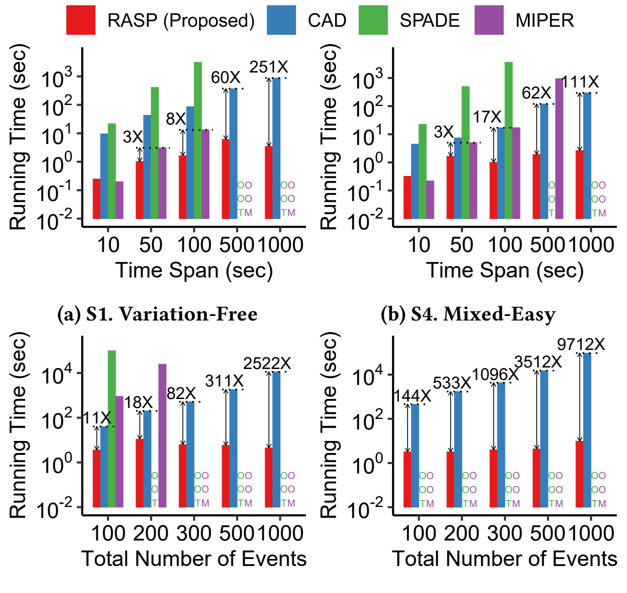
Figure 5: Q1. Accuracy. Our proposed method, RASP, exhibits superior accuracy compared to its competitors across all tested settings (S1-S5). OOT: out-of-time (> 100k seconds). OOM: out-of-memory.

5, 10, 20, and 100, the choice of the metric or the value of n did not significantly affect the relative results. Thus, we reported NDCG@5 values throughout Sect. 5, aligning with the number of ground-truth TSPs in our datasets, and the results in terms of NDCG@20 and RC@5 are additionally reported in Section C of the online appendix [8]. In cases of ties, we assigned fractional scores to tied TSPs, as in [32]. We computed the metrics based only on the event composition in output TSPs, while ignoring time gaps, to directly compare different methods based on slightly different notions of (relaxed) TSPs.

5.2 Q1. Accuracy

We assessed the accuracy of the considered methods under the settings described in Sect. 5.1 (i.e., S1-S5). The results are given in Fig. 5. Our observations can be summarized as follows:

- RASP consistently outperformed all its competitors across all the considered settings.
- RASP achieved the same level of accuracy as its competitors with a shorter time span (i.e., a smaller amount of input data). Note that in real-world scenarios, collecting datasets over extended periods is often challenging and costly.
- Temporal variations can cause the supports of ground-truth TSPs to be underestimated, making the discovery of ground-truth TSPs more challenging, especially when the number of events grows and thus the number of frequent TSPs grows (see (d) and (e) in Fig. 5). This issue is addressed in RASP by introducing relaxed TSPs but not in other methods.
- The relative advantage of RASP became more evident in more challenging settings. For instance, compare (b) and (c), (d) and (e), and (f) and (g) in Fig. 5.
- In the most challenging setting (**S5. Mixed-Hard**), only RASP attained a satisfactory level of accuracy.
- SPADE and MIPER showed limited scalability with extended time spans or more events, which leads to an increased number of frequent TSPs. However, RASP uses a resource-adaptive mechanism to keep the number of frequent TSPs considered manageable, preventing scalability issues (out-of-memory or out-of-time).



(a) S1. Variation-Free

(b) S4. Mixed-Easy

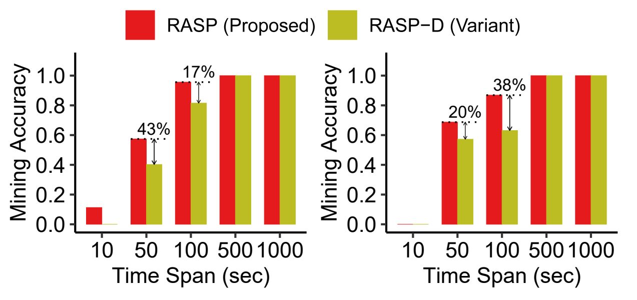
(c) S3. Event Count (time span: 100 secs)

(d) S3. Event Count (time span: 1, 000 secs)

Figure 6: Q2. Scalability. The running time of RASP grows gradually with the time span of data and the number of events, in contrast to the other competing methods.

5.3 Q2. Scalability

In order to evaluate the scalability of the considered methods, we examined their running times under various settings. The results are given in Fig. 6 and summarized as follows:



(a) S2. Variations (20 ms) **(b) S4. Mixed-Easy**

Figure 7: Q4. Ablation study: (1) effectiveness of duplicated pattern matching using relaxed TSPs. RASP performs better or equally well than a variant without this feature, demonstrating its effectiveness.

- RASP was the fastest in almost all settings.
- Its running time grew gradually with the time span of data and the number of events, in contrast to other competing methods.
- RASP was up to 9, 712× faster than the competitor in the largest dataset with 1, 000 events and a time span of 1, 000 seconds.

The superior scalability of RASP is attributed largely to resource-adaptive thresholding (refer to Sect. 4.4). It maintains the number of TSPs at a feasible level regardless of the input data size.

5.4 Q3. Speed-Accuracy Trade-off

We examined the trade-offs between the accuracy and speed of the considered methods, which are controlled by varying the time span of the input data. The results for two settings, **S1. Variation-Free** and **S4. Mixed-Easy**, are given in Figs. 1(a) and (b). In both settings, RASP provided clearly better trade-offs than the other methods, as expected from the previous results.

5.5 Q4. Ablation Studies

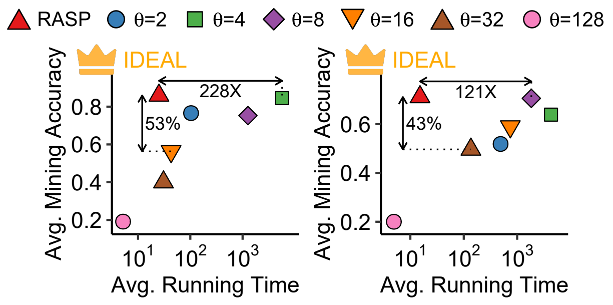
In this subsection, we conducted ablation studies to verify the effectiveness of the key ideas behind RASP.

Firstly, we assessed the effectiveness of duplicated pattern matching using relaxed TSPs (described in Sect. 4.3) by comparing RASP and a variant without this feature (RASP-D). As shown in Fig. 7, RASP consistently performed better or equally well compared to RASP-D, demonstrating its effectiveness.

Secondly, we examined the effectiveness of resource-adaptive automatic hyperparameter tuning (described in Sect. 4.4) by comparing RASP and variants with thresholds fixed to predefined values. We measured accuracy (in terms of NDCG@5) and speed averaged over all time spans (i.e., 10, 50, 100, 500, and 1000 secs)[7] As shown in Fig. 8, RASP with resource-adaptive automatic hyperparameter tuning provides a significantly better trade-off between accuracy and speed compared to the variants with fixed thresholds, demonstrating its effectiveness.

Lastly, we evaluated the space reduction of the tree-based structure (described in Sect. 4.5) of RASP by measuring the reduction in the number of nodes due to the re-use of intermediate

[7]The variants ran out of memory (OOM) under the following conditions: (1) when the threshold θ was set to 2 and the time spans were 500 and 1,000 seconds in the **S1. Variation-Free** setting; and (2) when θ was set to 4 and the time span was 1,000 seconds in the **S4. Mixed-Easy** setting. Thus, the results in these settings were disregarded when calculating the performance of the variants. Note that the average running times of the variants would have further increased if their running times in these settings had been considered.



(a) S1. Variation-Free **(b) S4. Mixed-Easy**

Figure 8: Q4. Ablation study: (2) effectiveness of resource-adaptive automatic hyperparameter tuning. RASP provides a significantly better trade-off between accuracy and speed than its variants with fixed thresholds.

Table 3: Q4. Ablation study: (3) space reduction (in %) by the tree-based data structure.

Time Span (sec)	10	50	100	500	1000	**Avg.**
S1. Variation-Free	67.8%	68.4%	70.2%	67.7%	68.3%	**68.5%**
S2. Variations (10ms)	71.8%	71.1%	72.2%	70.1%	68.7%	**70.8%**
S2. Variations (20ms)	71.7%	71.0%	72.1%	70.0%	68.9%	**70.7%**
S4. Mixed-Easy	72.0%	72.4%	71.2%	70.2%	68.7%	**70.9%**
S5. Mixed-Hard	71.5%	72.0%	71.8%	70.1%	68.2%	**70.7%**

nodes in the prefix trees. That is, we compared the total number of nodes in the prefix trees to the sum of pattern length ($\sum_i$ pattern length i×number of patterns of length i), i.e. the number of nodes if we do not employ the re-use of intermediate nodes. As shown in Table 3, the tree-based structure significantly reduces the space requirement, demonstrating its effectiveness.

5.6 Extra Results on Real-World Datasets without Ground-Truth TSPs

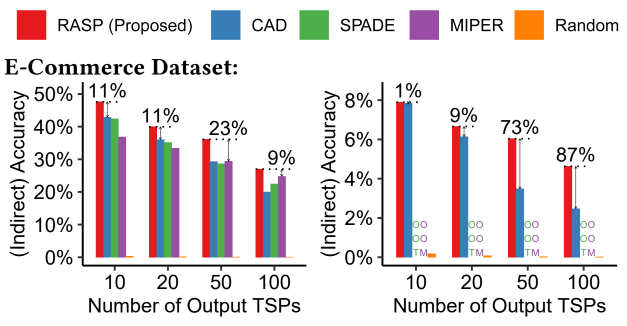
Ideally, for evaluation, the event composition and time gaps of the ground-truth TSPs and their instances should be available, despite temporal variations. Unfortunately, we could not find suitable real-world datasets that meet these criteria. Hence, in this subsection, we evaluate the considered methods indirectly using four real-world datasets without ground-truth TSPs.

5.6.1 E-commerce. The e-commerce dataset consists of timestamped event instances, with each event indicating a click on a specific product. This dataset naturally includes temporal variations as different users have varying click speeds.

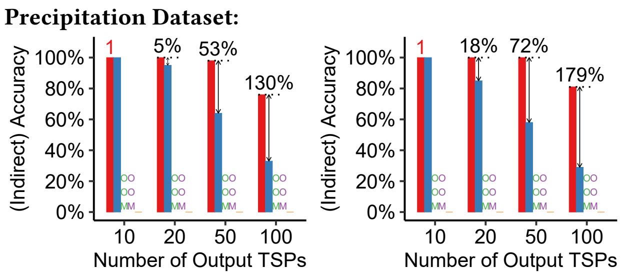
Details of the Dataset. We used an e-commerce dataset obtained from YOOCHOOSE GmbH, an online retailer[8]. The dataset was collected over several months in 2014 and consists of temporal click events. Each record in the dataset includes a session ID, an occurrence time of a temporal event instance, and an event ID, which indicates a click on a specific product. Note that session IDs were used solely for evaluation purposes and not as input.

In our experiment, we focused on the temporal events that took place on April 27, a day with a high volume of temporal events. To ensure the practical running time and memory usage of the competing methods, we limited our analysis to the 100 most frequent temporal events. We converted the timestamps from

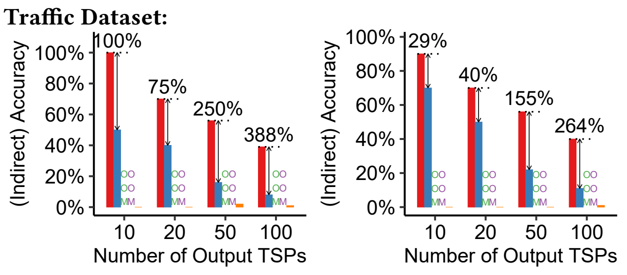
[8]https://www.kaggle.com/datasets/chadgostopp/recsys-challenge-2015



(a) Bin size: 1 min (b) Bin size: 1 sec



(c) Bin size: 5 min (d) Bin size: 15 min



(e) Bin size: 15 min (f) Bin size: 30 min

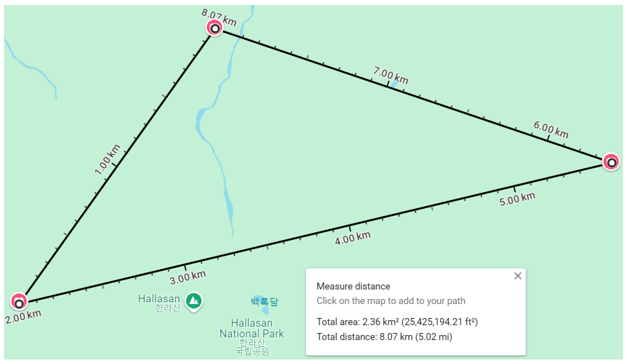
Figure 9: Extra results on three real-world datasets (e-commerce, precipitation, and traffic) without ground-truth TSPs. RASP consistently achieves the highest accuracy measured in an indirect manner (see Sect. 5.6.1-5.6.3).

milliseconds to minutes or seconds by binning them into one-minute or one-second intervals, respectively. This preprocessing results in 165, 118 event instances and 53, 658 sessions.

Experimental Setups. In our experiment, we set the maximum time span of a TSP L to 600 seconds, which is more than twice the average time span per session without limiting the maximum time gap δ. All other settings remained consistent with those used in the main experiments. The tolerance against temporal variations I was set to 1 minute and 5 seconds for a bin size of 1 minute and 1 second, respectively. As an additional baseline, we considered **RANDOM**, where TSPs were composed of events chosen uniformly at random. We applied the considered methods to extract frequent TSPs of length 3. As ground-truth TSPs are not available, we evaluated their accuracy through an indirect method. Specifically, for each output TSP instance,[9] we measured the proportion where all event instances within it originated from a single web session. It is important to note that event instances from a single web session are more likely to be related (and thus meaningful) than instances from multiple sessions, which may involve different users.

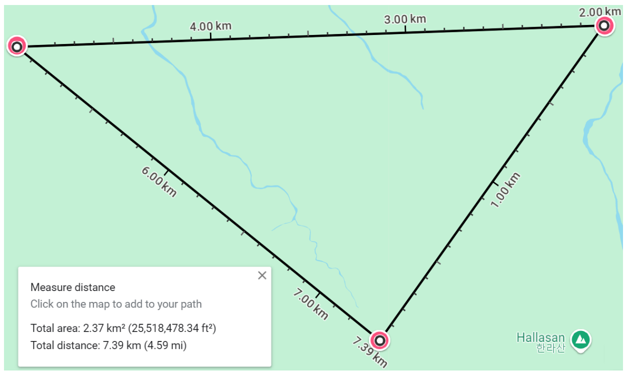
Results. Figs. 9(a)-(b) show the results with varying time-bin sizes and output TSP counts. RASP consistently performed best.

[9]For evaluation, after obtaining output TSPs, we merged ones with the same event composition but different time gaps into a single TSP. This is required to directly compare different methods based on slightly different notions of (relaxed) TSPs.



Measure distance
Click on the map to add to your path
Total area: 2.36 km² (25,425,194.21 ft²)
Total distance: 8.07 km (5.02 mi)

(a) Most Frequent TSP



Measure distance
Click on the map to add to your path
Total area: 2.37 km² (25,518,478.34 ft²)
Total distance: 7.39 km (4.59 mi)

(b) 2nd Most Frequent TSP

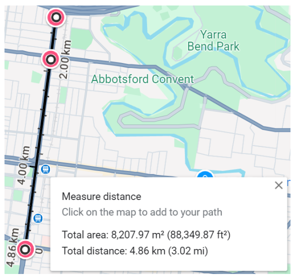
Figure 10: The AWS locations for the two most frequent TSPs identified by RASP from the precipitation dataset. Note that the AWSs are closely located.

5.6.2 *Precipitation.*

The precipitation dataset is a time series of event instances, with each event indicating precipitation observed at each automated weather station (AWS).

Details of the Dataset. The data was collected from 714 AWS stations across South Korea, recording precipitation accumulated over 15-minute intervals with measurements taken every 5 minutes. Each entry includes the timestamp, the amount of accumulated precipitation, and the corresponding AWS ID. To ensure manageable running time and memory usage for the competing methods, we focused on temporal events in 2020 (i.e., the year with the most precipitation in South Korea between 2018 and 2022) collected from the 100 AWSs with the highest average precipitation levels, yielding 1, 111, 827 event instances.

Experimental Setups. In our experiments, we defined each temporal event as precipitation (accumulated over the past 15 minutes) of 1.0 mm or more observed at each AWS. We set the maximum time span of a TSP L to 1 hour and the tolerance for temporal variations I to values of 1, 2, and 3 bins, which are either 5 or 15 minutes. All other settings were consistent with those used in Sect. 5.6.1. Since ground-truth TSPs were unavailable, we evaluated the accuracy of the identified TSPs using an indirect way. Specifically, we measured the percentage of output TSPs where the average distance between the AWSs within the TSP is reasonably close (spec., 5 km or less).



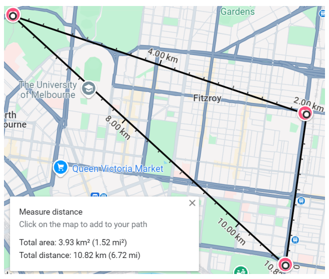
Measure distance

Click on the map to add to your path

Total area: 8,207.97 m² (88,349.87 ft²)

Total distance: 4.86 km (3.02 mi)

(a) Most Frequent TSP



Measure distance

Click on the map to add to your path

Total area: 3.93 km² (1.52 mi²)

Total distance: 10.82 km (6.72 mi)

(b) 2nd Most Frequent TSP

Figure 11: The sensor locations for the two most frequent TSPs identified by RASP from the traffic dataset Note that the sensors are closely located.

<u>**Results.**</u> Figs. 9(c)-(d) presents the results with varying time bin sizes and output TSP counts. RASP consistently performed best. See Figure 10 for the two most frequent TSPs identified by RASP.

5.6.3 *Traffic.*

The traffic dataset is a time series of event (spec., traffic congestion detected at each sensor) instances.

<u>**Details of the Dataset.**</u> We used data collected from 1,084 sensors embedded under the roads in Melbourne and its surrounding suburbs [43]. The dataset includes traffic volume measured every 15 minutes. Each record contains the timestamp, traffic volume, and the corresponding sensor ID. We focused on the temporal event instances in 2011 (i.e., the year with the highest traffic volume between 2007 and 2012) collected from the 100 sensors recording the highest traffic volumes, resulting in $1,080,579$ event instances. Note that, even with this reduction, most competing methods are still not scalable enough to handle the data.

<u>**Experimental Setups.**</u> We defined each temporal event as a traffic volume of 500 vehicles or more (over the past 15 minutes) recorded at each sensor. We set the maximum time span of a TSP L to 3 hours, and all other settings were consistent with those used in Sect. 5.6.2. We evaluated the accuracy of the identified TSPS, indirectly, by the ratio of output TSPs where the distance sum among the sensors within each TSP is reasonably small (spec., at most 4 km).

<u>**Results.**</u> Figs. 9(e)-(f) present the results, where RASP consistently performed best regardless of time bin sizes and output TSP counts. See Fig. 11 for the two most frequent TSPs from RASP.

5.6.4 *Stock.*

The stock dataset is a time series of event instances, where each event represents a significant fluctuation in the daily price of each stock.

<u>**Details of the Dataset.**</u> We used data from Investing.com and Yahoo Finance [27]. The dataset spans from January 3, 2008, to June 30, 2020, and includes daily prices and trading volumes for 3,028 stocks, along with information on the stock names and sectors. Among them, we focused on the 101 stocks with a daily average trading value of 10 billion Korean won or more. This preprocessing yields 17,040 event instances.

<u>**Experimental Setups.**</u> We defined each temporal event as a significant fluctuation (spec., daily return rate exceeding 5% or dropping below -5%) of the price of each stock. We set the maximum time span L of a TSP to 20 days, with a bin size of 1 day. All other settings were consistent with those in Sect. 5.6.2.

<u>**Results.**</u> The frequent TSPs identified by RASP capture sector- and affiliate-based relationships, as illustrated below.

(1) **Sector-Based Patterns**: Many identified TSPs (e.g., the two most frequent ones) involve stocks from the same sector.
   - The most frequent TSP (with a support of 84) consists of (1) GS Engineering & Construction Corporation, (2) DL Holdings, and (3) Hyundai Engineering & Construction, all of which belong to the Engineering & Construction sector.
   - The second most frequent TSP (with a support of 82) consists of (1) Hyundai Mipo Dockyard, (2) Samsung Heavy Industries, (3) Korea Shipbuilding & Offshore Engineering, all of which belong to the Shipbuilding & Offshore Engineering sector.

(2) **Affiliate-Based Patterns.** Some of the identified TSPs, including the third most frequent one, consist of stocks from the same corporate affiliates.
   - The third most frequent TSP (with a support of 75) consists of (1) Doosan Infracore, (2) Doosan Heavy Industries & Construction, and (3) Doosan Corporation, all of which are part of the Doosan group.

6 CONCLUSION AND FUTURE DIRECTIONS

In this paper, we focused on the problem of mining frequent TSPs in a sequence of temporal events with temporal variations. For the problem, we proposed RASP, which incorporates (a) a novel concept of relaxed TSPs for handling variations, (b) resource-adaptive automatic hyperparameter tuning for enhancing usability, and (c) a tree-based concise data structure for space efficiency. Our experimental results demonstrated that RASP exhibited substantial improvements in accuracy and robustness (see Figs. 5 and 9); and speed and scalability (see Fig. 6); compared to four state-of-the-art competitors. Moreover, to achieve the same level of accuracy, RASP demanded a smaller amount of data (see Fig. 5). Lastly, we leave the incorporation of additional criteria for TSPs (e.g., closedness and maximality) into RASP for future work to better identify the most relevant and non-redundant TSPs.

ACKNOWLEDGMENTS

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2024-00438638, EntireDB2AI: Foundations and Software for Comprehensive Deep Representation Learning and Prediction on Entire Relational Databases, 90%) (No. RS-2019-II190075, Artificial Intelligence Graduate School Program (KAIST), 10%).

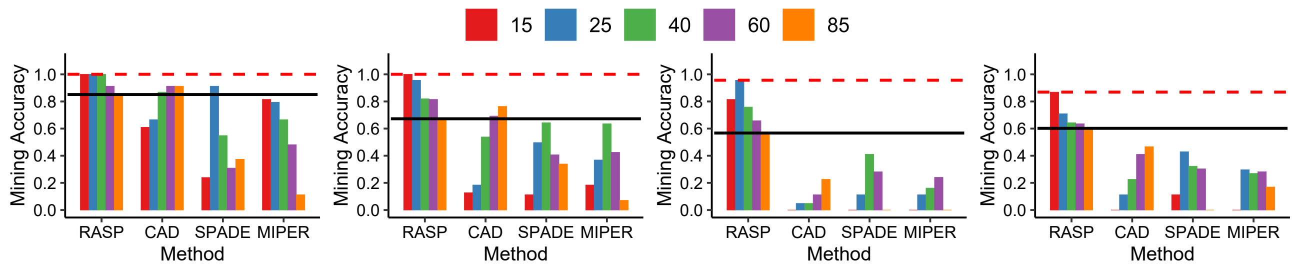
APPENDIX

A ADDITIONAL EXPERIMENTAL RESULTS

Below, we present the results of additional experiments.

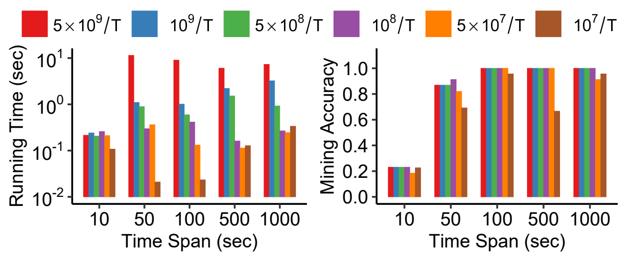
A.1 Effects of M (i.e., Maximum Number of TSPs to Retain)

The maximum number M of TSPs to retain needs to be specified as a hyperparameter considering the total memory size and the memory size per TSP. To investigate the effect of M, we varied its value in 6 cases and measured the running time and accuracy. As depicted in Fig. 13, under a setting of **S2. Variations** (10 ms), we observed that both the running time and mining accuracy of RASP tend to decrease as M decreases. While prioritizing accuracy over speed, we fixed the value of M at $10^9/T$, where T is the time span, for all other experiments.



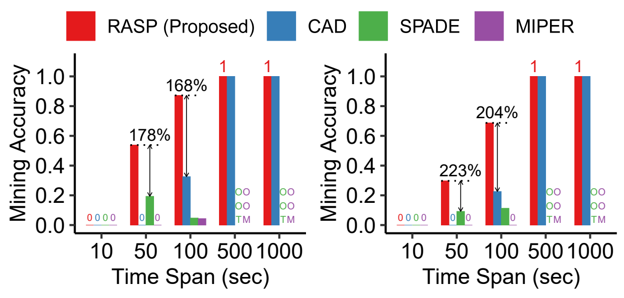
(a) S1. Variation-Free **(b) S2. Variations (10 ms)** **(c) S2. Variations (20 ms)** **(d) S4. Mixed-Easy**

Figure 12: Results with various bin sizes in milliseconds. Red dotted lines and black solid line show the best and worst performances of RASP. Note that larger time bins tend to benefit CAD, but not necessarily the other methods for which the optimal bin size tends to increase with larger temporal variations. RASP performs best in most cases regardless of bin sizes.

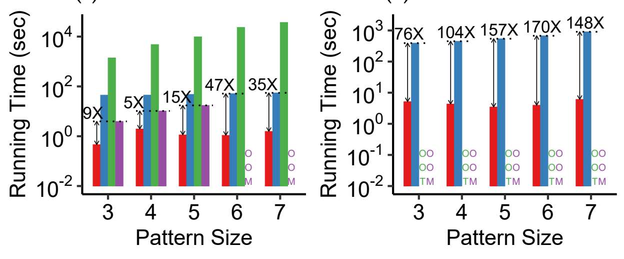


(a) Running time **(b) Mining accuracy**

Figure 13: Effects of the max number M of TSPs to retain. Under S2. Variations (10 ms), the running time and mining accuracy of RASP tend to decrease as M decreases.



(a) Pattern Size: 6 **(b) Pattern Size: 7**



(c) Vs. Pattern size (time span: 100 secs) **(d) Vs. Pattern size (time span: 1, 000 secs)**

Figure 14: Results with TSPs of various sizes. Our proposed method, RASP, outperforms all its competitors in terms of both speed and accuracy, regardless of the TSP size.

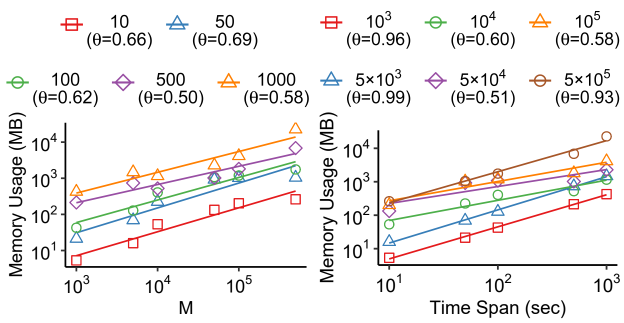


Figure 15: Empirical memory usage of RASP, which grows (sub-)linearly with M and T. In the legend, θ indicates the slope of the fitted regression line.

A.2 Results with TSPs of Various Sizes

While varying the size of ground-truth TSPs in the input data, we measured the accuracy and running times of the considered methods under the **S4. Mixed-Easy** setting. The results are given in Fig. 14, and our observations are summarized as follows:

- RASP outperformed all its competitors in terms of both speed and accuracy, regardless of the TSP size.
- The relative superiority of RASP in accuracy was more evident in more challenging settings (compare (a) and (b) in Fig. 14).

A.3 Results with Various Bin Sizes

While varying the size of bins in the input data, we measured the accuracy of the considered methods under four different settings. From the results in Fig. 12, where we set the time span T to 100 seconds, we made the following observations.

- Larger time bins tended to benefit CAD.
- For the other methods, the optimal bin size tended to increase with larger temporal variations.
- RASP performed best in most cases regardless of bin sizes.

A.4 Empirical Memory Usage

We measured the empirical memory usage of RASP under the **S2. Variations** (10 ms) setting while varying the time span T and the budget M of TSPs. As shown in Fig. 15, the memory usage grew (sub-)linearly with M and T, consistently with our theoretical complexity $O(M \cdot T)$. Note that the theoretical complexity of $M \cdot T$ represents a worst-case scenario, and the actual trend is often sub-linear rather than strictly linear. That is, the slopes of the fitted regression lines on the log-log scale (refer to the θ values in the legend) are often much smaller than 1.

REFERENCES

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining association rules between sets of items in large databases. In *SIGMOD*.

[2] Rakesh Agrawal and Ramakrishnan Srikant. 1995. Mining sequential patterns. In *ICDE*.

[3] Rakesh Agrawal, Ramakrishnan Srikant, et al. 1994. Fast algorithms for mining association rules. In *VLDB*.

[4] Xiang Ao, Ping Luo, Jin Wang, Fuzhen Zhuang, and Qing He. 2018. Mining precise-positioning episode rules from event sequences. *IEEE Transactions on Knowledge and Data Engineering* 30, 3 (2018), 530–543.

[5] Christian Borgelt. 2012. Frequent item set mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2, 6 (2012), 437–456.

[6] Sergey Brin, Rajeev Motwani, Jeffrey D Ullman, and Shalom Tsur. 1997. Dynamic itemset counting and implication rules for market basket data. In *SIGMOD*.

[7] Gemma Casas-Garriga. 2003. Discovering unbounded episodes in sequential data. In *PKDD*.

[8] Hyunjin Choo, Minho Eom, Gyuri Kim, Young-Gyu Yoon, and Kijung Shin. 2024. Online appendix of 'RASP: Robust Mining of Frequent Temporal Sequential Patterns under Temporal Variations'. https://github.com/jin-choo/RASP/blob/master/appendix.pdf.

[9] Bolin Ding, David Lo, Jiawei Han, and Siau-Cheng Khoo. 2009. Efficient mining of closed repetitive gapped subsequences from a sequence database. In *ICDE*.

[10] Min Fang, Narayanan Shivakumar, Hector Garcia-Molina, Rajeev Motwani, and Jeffrey D Ullman. 1999. Computing Iceberg Queries Efficiently.. In *VLDB*.

[11] Philippe Fournier-Viger, Antonio Gomariz, Ted Gueniche, Azadeh Soltani, Cheng-Wei Wu, and Vincent S Tseng. 2014. SPMF: A Java Open-Source Pattern Mining Library. *Journal of Machine Learning Research* 15, 104 (2014), 3569–3573.

[12] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Bay Vo, Tin Truong Chi, Ji Zhang, and Hoai Bac Le. 2017. A survey of itemset mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 7, 4 (2017), e1207.

[13] Jiaqi Ge, Yuni Xia, and Jian Wang. 2015. Towards efficient sequential pattern mining in temporal uncertain databases. In *PAKDD*.

[14] Bart Goethals. 2003. Survey on frequent pattern mining. *Univ. of Helsinki* 19 (2003), 840–852.

[15] Antonio Gomariz, Manuel Campos, Roque Marin, and Bart Goethals. 2013. Clasp: An efficient algorithm for mining frequent closed sequences. In *PAKDD*.

[16] Şule Gündüz and M Tamer Özsu. 2003. A web page prediction model based on click-stream tree representation of user behavior. In *KDD*.

[17] Thomas Guyet and René Quiniou. 2008. Mining temporal patterns with quantitative intervals. In *ICDM Workshops*.

[18] Thomas Guyet and René Quiniou. 2011. Extracting temporal patterns from interval-based sequences. In *IJCAI*.

[19] Thomas Guyet and René Quiniou. 2020. NegPSpan: efficient extraction of negative sequential patterns with embedding constraints. *Data Mining and Knowledge Discovery* 34 (2020), 563–609.

[20] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. 2007. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery* 15 (2007), 55–86.

[21] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Meichun Hsu. 2001. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *ICDE*.

[22] Jiawei Han, Jian Pei, and Yiwen Yin. 2000. Mining frequent patterns without candidate generation. In *SIGMOD*.

[23] Mike Hemberger, Mark Shein-Idelson, Lorenz Pammer, and Gilles Laurent. 2019. Reliable sequential activation of neural assemblies by single pyramidal cells in a three-layered cortex. *Neuron* 104, 2 (2019), 353–369.

[24] Yu Hirate and Hayato Yamana. 2006. Generalized sequential pattern mining with item intervals. *J. Comput.* 1, 3 (2006), 51–60.

[25] Kuo-Yu Huang and Chia-Hui Chang. 2008. Efficient mining of frequent episodes from complex sequences. *Information Systems* 33, 1 (2008), 96–114.

[26] Yuji Ikegaya, Gloster Aaron, Rosa Cossart, Dmitriy Aronov, Ilan Lampl, David Ferster, and Rafael Yuste. 2004. Synfire chains and cortical songs: temporal modules of cortical activity. *Science* 304, 5670 (2004), 559–564.

[27] Jun-Gi Jang and U Kang. 2021. Fast and Memory-Efficient Tucker Decomposition for Answering Diverse Time Range Queries. In *KDD*.

[28] Xiaonan Ji, James Bailey, and Guozhu Dong. 2007. Mining minimal distinguishing subsequence patterns with gap constraints. *Knowledge and Information Systems* 11 (2007), 259–286.

[29] Somayah Karsoum, Le Gruenwald, Clark Barrus, and Eleazar Leal. 2019. Using timed sequential patterns in the transportation industry. In *Big Data*.

[30] Srivatsan Laxman, P Sastry, and K Unnikrishnan. 2007. Discovering frequent generalized episodes when events persist for different durations. *IEEE Transactions on Knowledge and Data Engineering* 19, 9 (2007), 1188–1201.

[31] Heikki Mannila, Hannu Toivonen, and A Inkeri Verkamo. 1997. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery* 1 (1997), 259–289.

[32] Frank McSherry and Marc Najork. 2008. Computing information retrieval performance measures efficiently in the presence of tied scores. In *ECIR*.

[33] Iris Miliaraki, Klaus Berberich, Rainer Gemulla, and Spyros Zoupanos. 2013. Mind the gap: Large-scale frequent sequence mining. In *SIGMOD*.

[34] Muhammad Muzammal and Rajeev Raman. 2010. On probabilistic models for uncertain sequential pattern mining. In *ADMA*.

[35] Mirco Nanni and Christophe Rigotti. 2006. Extracting trees of quantitative serial episodes. In *KDID*.

[36] Anny Ng and Ada Wai-chee Fu. 2003. Mining frequent episodes for relating financial events and stock trends. In *PAKDD*.

[37] Jong Soo Park, Ming-Syan Chen, and Philip S Yu. 1995. An effective hash-based algorithm for mining association rules. *ACM Sigmod Record* 24, 2 (1995), 175–186.

[38] Gregory Piatetsky-Shapiro. 1991. Discovery, analysis, and presentation of strong rules. *Knowledge Discovery in Data-bases* (1991), 229–248.

[39] Pietro Quaglio, Alper Yegenoglu, Emiliano Torre, Dominik M Endres, and Sonja Grün. 2017. Detection and evaluation of spatio-temporal spike patterns in massively parallel spike train data with spade. *Frontiers in Computational Neuroscience* 11 (2017), 41.

[40] Eleonora Russo and Daniel Durstewitz. 2017. Cell assemblies at multiple time scales with arbitrary lag constellations. *Elife* 6 (2017), e19428.

[41] PS Sastry and KP Unnikrishnan. 2010. Conditional probability-based significance tests for sequential patterns in multineuronal spike trains. *Neural Computation* 22, 4 (2010), 1025–1059.

[42] Ashoka Savasere, Edward Omiecinski, and Shamkant Navathe. 1995. An E cient Algorithm for Mining Association Rules in Large Databases. In *VLDB*.

[43] Florin Schimbinschi, Xuan Vinh Nguyen, James Bailey, Chris Leckie, Hai Vu, and Rao Kotagiri. 2015. Traffic forecasting in complex urban networks: Leveraging big data and machine learning. In *Big Data*.

[44] Bai-En Shie, Hui-Fang Hsiao, and Vincent S Tseng. 2013. Efficient algorithms for discovering high utility user behavior patterns in mobile commerce environments. *Knowledge and Information Systems* 37 (2013), 363–387.

[45] Ramakrishnan Srikant and Rakesh Agrawal. 1996. Mining sequential patterns: Generalizations and performance improvements. In *EDBT*.

[46] Alessandra Stella, Pietro Quaglio, Emiliano Torre, and Sonja Grün. 2019. 3d-SPADE: Significance evaluation of spatio-temporal patterns of various temporal extents. *Biosystems* 185 (2019), 104022.

[47] Hannu Toivonen. 1996. Sampling large databases for association rules. In *VLDB*.

[48] Emiliano Torre, David Picado-Muiño, Michael Denker, Christian Borgelt, and Sonja Grün. 2013. Statistical evaluation of synchronous spike patterns extracted by frequent item set mining. *Frontiers in Computational Neuroscience* 7 (2013), 132.

[49] Takeaki Uno, Masashi Kiyomi, Hiroki Arimura, et al. 2004. LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *FIMI*.

[50] Jianyong Wang and Jiawei Han. 2004. BIDE: Efficient mining of frequent closed sequences. In *ICDE*.

[51] Alex Williams, Anthony Degleris, Yixin Wang, and Scott Linderman. 2020. In Point process models for sequence detection in high-dimensional neural spike trains. *NeurIPS*.

[52] Xifeng Yan, Jiawei Han, and Ramin Afshar. 2003. Clospan: Mining: Closed sequential patterns in large datasets. In *SDM*.

[53] Mohammed J Zaki. 2001. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning* 42 (2001), 31–60.

[54] Chao Zhang, Jiawei Han, Lidan Shou, Jiajun Lu, and Thomas La Porta. 2014. Splitter: Mining fine-grained sequential patterns in semantic trajectories. *PVLDB* 7, 9 (2014), 769–780.