Highlights

VeTraSPM: Novel Vehicle Trajectory Data Sequential Pattern Mining Algorithm for Link Criticality Analysis

Nourhan Bachir,Chamseddine Zaki,Hassan Harb,Roland Billen

- Introduces VeTraSPM, a novel algorithm for mining vehicle trajectory data using sequential pattern mining, optimized for dynamic vehicle movement patterns.
- Overcomes limitations of existing sequential data mining algorithms by incorporating directional constraints, repetitive sequences, and traffic network structures specific to vehicle trajectory data.
- Proposes the Sequential Impact Score (SIS), a novel metric that quantitatively evaluates road link criticality, improving traffic resilience analysis.
- Showcases the computational efficiency and scalability of VeTraSPM through partitioning and parallel processing, ensuring its applicability to large-scale urban traffic networks.
- Validates VeTraSPM through a real-world case study, demonstrating its effectiveness in identifying critical road links for improved traffic management and resilience planning.

VeTraSPM: Novel Vehicle Trajectory Data Sequential Pattern Mining Algorithm for Link Criticality Analysis

Nourhan Bachir[a,*], Chamseddine Zaki[b], Hassan Harb[b] and Roland Billen[a]

[a]*Geomatics Unit, Faculty of Sciences, Universite de Liège, Liège, Belgium*
[b]*College of Engineering and Technology, American University of the Middle East, Kuwait*

ARTICLE INFO

Keywords:
Critical Link Analysis
Data Mining
Sequential Pattern Mining
Vehicle Trajectory Data
Intelligent Traffic Management

ABSTRACT

This paper presents VeTraSPM (Vehicle Trajectory Data Sequential Pattern Mining), a novel algorithm designed to address the limitations of existing sequential pattern mining methods when applied to vehicle trajectory data. Current algorithms fail to capture essential characteristics such as directional flow on one-way roads (e.g., "AB" is valid but not "BA"), connectivity constraints at junctions, and the repetition of links within sequences. VeTraSPM overcomes these gaps by accurately extracting frequent patterns and confident rules while leveraging vertical projection for efficient memory and space management, enabling it to handle large datasets. Furthermore, the algorithm incorporates partitioning and parallelization techniques, further enhancing its scalability for real-world traffic environments. Three new metrics—FqMS, CMS, and SIS—are introduced to assess link criticality based on the consistent occurrence of links across movement patterns at various levels. The efficiency of VeTraSPM is demonstrated through a comparative analysis with baseline algorithms, showcasing its superior performance. The visualization of the proposed metrics offers valuable insights into link importance, supporting proactive traffic management strategies. A case study using real-world datasets from Luxembourg and Monaco validates its scalability and practical value in enhancing the resilience of urban traffic networks.

1. Introduction

Transportation networks are essential lifelines that enable the movement of people, goods, and services, contributing to economic growth, social connectivity, and overall well-being. However, these networks often face disruptions due to accidents, natural disasters, construction, or other unforeseen events, requiring transportation authorities to manage traffic efficiently and mitigate the impact of such disruptions. Identifying critical links—key segments of the network where disruptions cause significant traffic bottlenecks—has become vital to supporting resilient transportation systems. Traditional approaches rely on static attributes and fail to account for the dynamic nature of vehicle trajectories, limiting their ability to assess link criticality accurately.

Sequential pattern mining is an effective technique to analyze temporal data and discover meaningful patterns from sequences. However, when applied to vehicle trajectory data, existing algorithms encounter several limitations. Firstly, they do not accommodate directional constraints in one-way roads, where, for example, "AB" might be a valid sequence, but "BA" is not. Secondly, they fail to capture the connectivity constraints among links, meaning only certain links can follow each other based on the road network's structure (e.g., linked by a junction). Thirdly, the potential repetition of links (e.g., a vehicle revisiting the same location) is often ignored, leading to inaccurate pattern detection in complex urban traffic networks. These limitations restrict the utility of existing algorithms in real-world scenarios.

To address these challenges, the introduced VeTraSPM (Vehicle Trajectory Data Sequential Pattern Mining), a novel algorithm tailored to the characteristics of vehicle trajectory data. VeTraSPM accurately discovers frequent movement patterns and confident rules, considering directionality, connectivity, and repetition. The algorithm employs vertical projection techniques, allowing for efficient memory and space management, enabling it to handle large datasets. Additionally, VeTraSPM incorporates partitioning and parallelization strategies to further enhance scalability, making it practical for real-world large-scale urban traffic systems. These optimizations ensure the algorithm's efficiency even with extensive datasets and multiple cores, improving both time and space performance during pattern mining.

nbachir@uliege.be (N. Bachir); chamseddine.zaki@aum.edu.kw (C. Zaki); hassan.harb@aum.edu.kw (H. Harb); rbillen@uliege.be (R. Billen)
ORCID(s):

In this work, three novel metrics—Frequent Movement Pattern Score (FqMS), Confident Movement Pattern Score (CMS), and Sequential Impact Score Index (SIS)—are proposed to assess link criticality based on the consistent appearance of links across frequent and confident movement patterns at various levels. The insights from these metrics are visualized to demonstrate how they can guide traffic managers in identifying critical links and making proactive decisions.

The experimentation in this study is conducted on the real-world transportation network of Luxembourg, incorporating traffic data from the SUMO LuST microscopic simulation scenario [1], the performance of VeTraSPM is compared against the baseline algorithms, demonstrating that VeTraSPM offers superior efficiency, especially when processing large datasets. LuST scenario is based on authentic traffic patterns incorporating genuine demographic data and activity demand through the ACTIVITYGEN tool. The evaluation dataset that was used for this scenario includes over six million Floating Car Data (FCD) samples collected in Luxembourg City as mentioned by the authors. With detailed metrics covering road networks, intersections, and various mobility aspects, using this scenario's data ensures a realistic assessment of the proposed methodology.

The organization of the paper is as follows: Section 2 provides an overview of related work in the field of vehicular data mining and analysis. Section 3 outlines the proposed methodology, explaining the new propositions, the application of association rules algorithm, and the determination of link criticality. Section 4 explains the implementation of the approach. Section 5 presents the experimentation and evaluation of the case study, comparing the performance of the proposed approach with existing methods. Finally, Section 6 concludes the paper, summarizing the contributions and highlighting avenues for future research.

This work offers new tools for urban planners and policymakers, facilitating more effective traffic management strategies in dynamic urban settings.

2. Related Works

This section provides an overview of the existing sequential data mining algorithms commonly used in trajectory data analysis. It also reviews various research works that have attempted to analyze critical links in transportation systems using different methodologies.

2.1. Existing Sequential Data Mining Algorithms

Frequent pattern mining plays a key role in sequential pattern discovery, with algorithms generally categorized into Apriori-based and FP-growth-based approaches. The Apriori algorithm [2] is widely used for association rule mining by identifying frequent itemsets and deriving rules based on support and confidence measures. However, its need for multiple database scans makes it computationally expensive and inefficient for large or complex datasets [3, 4, 5].

PrefixSpan [6] addresses these limitations by avoiding candidate generation, focusing instead on frequent subsequences, which reduces computational overhead. An enhanced version, PrefixSpan-x [7], further improves memory efficiency by pruning unnecessary patterns. Top-k sequence mining algorithms [8, 9, 10] discover the most relevant patterns by applying global constraints such as quantity or item relationships.

Constraint-based algorithms refine pattern discovery by introducing additional rules. These include regular expression constraints, weight-based constraints, and length constraints [11], which narrow the search space to improve relevance. Closed sequence mining [12, 13] eliminates redundant sequences, keeping only the most significant patterns. Incremental mining algorithms [14] update discovered patterns dynamically, avoiding full re-computation when new data becomes available.

Vertical data mining algorithms, such as SPAM [15], SPADE [16], CM-SPADE [17], ClaSP [18], and VMSP [19], offer further efficiency by converting datasets into vertical representations. These methods are particularly useful for large datasets, with SPAM leveraging depth-first search through item and sequence extensions to improve speed and memory usage. SPADE and CM-SPADE build on this by partitioning data into equivalence classes, further optimizing complex sequence mining.

Despite their efficiency, these vertical mining algorithms are not well-suited for vehicle trajectory data, which features ordered sequences, directional constraints, and repetition. Vehicle trajectories are continuous and may contain loops or revisits, patterns that these algorithms struggle to capture. This limitation reduces their effectiveness for critical link analysis in transportation systems, where such nuances are essential.

Table 1
Summary of Sequential Pattern Mining Algorithms and Techniques

Algorithm	**Category**	**Key Features and Limitations**
Apriori [2]	Apriori-based	Identifies frequent itemsets but requires multiple scans, leading to high time complexity.
PrefixSpan [6]	Prefix-based	Reduces search space but requires significant memory for long sequences.
PrefixSpan-x [7]	Enhanced Prefix-based	Optimizes memory by pruning unnecessary patterns.
Top-k Sequence Mining [8, 9, 10]	Constraint-based	Discovers top-k patterns with global constraints like quantity and item relations.
SPADE [16]	Vertical projection	Efficient for large datasets but struggles with ordered, repetitive sequences.
CM-SPADE [17]	Vertical mining	Partitions data into equivalence classes; memory-intensive for complex sequences.
ClaSP [18]	Class sequential mining	Identifies class-specific patterns but is limited by trajectory constraints.
SPAM [15]	Vertical bitmap-based	Uses depth-first search for efficient mining; limited for handling repetitive patterns in trajectories.
VMSP [19]	Vertical mining	Efficient for long sequences but lacks support for ordered and repetitive patterns.
Regular Expression Constraints [11]	Constraint-based	Uses regex, weight, and length constraints to enhance pattern relevance.
Weighted Sequence Mining [20]	Weighted patterns	Mines patterns with weights, improving relevance in specific contexts.
Closed Sequence Mining [12, 13]	Closed pattern discovery	Prunes redundant sequences, retaining only the most significant patterns.
Incremental Mining [14]	Incremental learning	Dynamically updates patterns but may struggle with large datasets.

Table 1 presents an overview of key sequential pattern mining algorithms and their characteristics. Although these algorithms are effective for various domains, they are limited in handling the specific properties of vehicle trajectory data, such as directionality, repetition, and sequence order.

Vehicle trajectory data requires specialized techniques that can accommodate its continuous nature and repetitive patterns. Conventional algorithms struggle with these features, reducing their applicability to transportation systems. Thus, despite the advancements in vertical mining algorithms, such as SPAM, SPADE, CM-SPADE, ClaSP, and VMSP, further improvements are needed to address the structural complexity of vehicle trajectories while maintaining computational efficiency.

2.2. Sequential Data Mining on Trajectory Data

Several studies have applied sequential pattern mining to vehicular data. Yu et al. [21] applied Apriori to discover frequent movement paths from taxi trajectories but encountered high time complexity due to repeated database scans. Ibrahim and Shafiq [22] combined clustering with the SPADE algorithm to generate insights from large taxi datasets, though their method struggled with ordered and repetitive patterns.

Wang et al. [23] used Apriori for planning flexible bus services by analyzing multiday path clusters but did not account for order and directionality. Hu et al. [24] applied frequent pattern mining to assess public transport stability but faced scalability issues. Moreira-Matias et al. [25] employed PrefixSpan to detect bus bunching, though their method overlooked repetitive sequences.

Table 2 presents key works in vehicular data analysis, demonstrating the limitations of basic algorithms in handling trajectory-specific challenges.

2.3. Sequential Data Mining for Critical Link Analysis

Despite its importance, critical link analysis using sequential pattern mining remains underexplored. Hu et al. [24] focused on public transport stability but did not address critical link identification. Zhang and He [26] explored

Table 2
Applications of Sequential Pattern Mining in Vehicular Data Analysis

Study	Algorithm Used	Limitations
Yu et al. [21]	Apriori	High time complexity, multiple scans.
Ibrahim and Shafiq [22]	SPADE	Struggles with order and repetition.
Wang et al. [23]	Apriori	Ignores road connectivity and directionality.
Hu et al. [24]	FP-growth	Limited scalability and prediction accuracy.
Moreira-Matias et al. [25]	PrefixSpan	High memory usage, overlooks order and repetitions.

Table 3
Applications of Sequential Mining in Critical Link Analysis

Study	Application	Focus and Limitations
Hu et al. [24]	Public transport stability	No focus on critical link analysis.
Zhang et al. [26]	VANET trajectory prediction	Focuses on movement paths, lacks critical link assessment.
Qi et al. [27]	Delay-tolerant networks	Timeliness-aware, lacks consideration of the order of links.
Merah et al. [28, 29]	Real-time tracking and route discovery	Route predition, relies on a basic implementation of sequential pattern mining.

trajectory prediction in VANETs without assessing critical links. Qi et al. [27] developed a timeliness-aware algorithm for delay-tolerant networks but their approach lacks consideration of the order of links, potentially affecting the quality of predictions. Merah et al. [28, 29] applied basic sequential mining techniques for real-time vehicle tracking and route discovery but lacked advanced mining methods to improve prediction accuracy.

Table 3 highlights the gaps in current research, emphasizing the need for an advanced approach that leverages association rules and sequential data mining techniques to identify critical links within transportation networks.

3. Vehicle Trajectories Sequential Pattern Mining (VeTraSPM)

An interesting approach for the discovery of critical links in transportation networks is the extraction of meaningful patterns from vehicle movement sequential data. The existing algorithms of sequence pattern discovery, like association rules algorithms such as Apriori suffer from their low accuracy when applied on vehicle trajectories data. In order to deal with this issue, this paper presents a new vehicle movement sequential data mining model for sequence pattern mining abbreviated *VeTraSPM* based on the vehicle trajectory data structure.

3.1. Preliminary Definitions

- *Trajectory Sequence (Tr):* $V = \{V_1, V_2, ..., V_n\}$ represents a group of vehicles that travel for a certain period in a given geographical area. $Tr_i = \{e_x, e_y, ..., e_n\}$ represents the trajectory sequence of vehicle i and $TrDB$ represents the set of sequences of all vehicles.

- *Sequence length:* The length of a sequence is simply the number of all links within this sequence. i.e. consider $Tr_k = \{e_x, e_y, e_z\}$ then $length(Tr_k) = length(\{e_x, e_y, e_z\}) = 3$.

- *Sub-sequence:* $Tr_i = \{e_x, e_y, \dots, e_n\}$, the trajectory sequence of vehicle i, is a sub-sequence of the trajectory sequence of vehicle j, $Tr_j = \{\dots, e_x, e_y, \dots, e_n, e_m, \dots\}$, where $length(Tr_i) \leq length(Tr_j)$ and Tr_j contains the whole sequence of Tr_i in the same order i.e. $\{e_x, e_y, e_z\}$ is a sub-sequence $\{e_l, e_x, e_y, e_z, e_m\}$, but not a sub-sequence of $\{e_x, e_l, e_y, e_m, e_z\}$

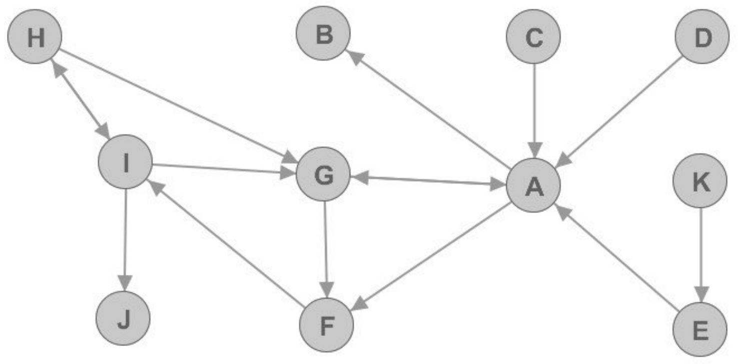

Figure 1: Example directed multi-graph

- *Movement pattern (Mp):* A movement pattern M represents a specific pattern (sub-sequence) to be detected in the trajectory data i.e $M = \{e_x, e_y\}$, $M' = \{e_z\}$.
- *Movement pattern order:* The order of a movement pattern is the length of the respective sequence. i.e. $order(M) = length(\{e_x, e_y\}) = 2$, $order(M') = length(\{e_z\}) = 1$. A movement pattern of order 1 is called “unit movement" which is in this case one edge/road/link like M'.
- *Movement rule:* A movement rule R is defined as association rule between two movement patterns expressed as $M \rightarrow M'$.
- *Support:* The support of movement pattern M is the number of appearances of this movement pattern as a sub-sequence in all trajectory sequences S. The support of the rule $R = M \rightarrow M'$, is the support of movement pattern MM' in the mobile database. i.e. $support(R) = support(MM') = support(\{e_x, e_y, e_z\})$
- *Confidence:* The confidence of movement rule R can be defined as: $confidence(R) = \frac{support(MM')}{support(M)}$.
- *Frequent Movement Pattern Set (FqM):* For a given support threshold *minsup*, a frequent movement pattern is a pattern whose support is not lower than *minsup*. A Frequent Movement Pattern Set denoted as FqM_k is the set of movement patterns of order k whose support is not lower than the *minsup* i.e. consider movement pattern $MM' = \{e_x, e_y, e_z\}$, if $support(\{e_x, e_y, e_z\}) \geq minsup$ then MM' is a frequent movement pattern and $MM' \in FqM_3$.
- *Confident Movement Pattern Set (CM):* For a given confidence threshold *minconf*, a confident movement pattern is a frequent movement pattern whose confidence is greater than or equal to *minconf*. For each FqM_k set of frequent movement patterns of order k, CM_k set of confident movement patterns can be extracted i.e. where $confidence(MM') = \frac{support(\{e_x,e_y,e_z\})}{support(\{e_x\})}$, if $confidence(MM') \geq minconf$, then MM' is a confident pattern and $MM' \in CM_3$.

Take the following example based on the graph in Fig. 1 and the trajectories database (TrDB) provided in Table 4. Let M be the movement pattern defining sequence $\{a, g\}$ and M' be the movement pattern defining sequence $\{f\}$. The confidence of trajectory association rules $R = M \rightarrow M'$ is the ratio of the number of trajectory sequences in the trajectory database that contain movement pattern MM', $support(MM')$, to the number of trajectory sequences that contain movement pattern M, $support(M) = support(\{a, g\}) = 3$. Consequently, $confidence(R) = support(MM')/support(M) = support(\{a, g, f\})/support(\{f\}) = 3/3 = 1$. A rule of confidence 1 is called a *perfect rule* which signifies that whenever the path M is taken M' is always the next step.

3.2. Vertical Projection of Trajectory Data

In order to come up with the frequent rules and the rules with best confidence, all the different combinations of items (edges) have to be explored and all the sequences in the database are queried over and over to find the support of each

Table 4
Example of vehicle driving trajectories database (TrDB)

Sequence (i)	Trajectory Seq. (Tr_i)	Sequence (i)	Trajectory Seq. (Tr_i)
1	[f, i, **g**, **a**]	8	[**g**, **a**, b]
2	[e, a, b]	9	[k, e, **a**, f, i, h, **g**]
3	[c, **a**, **g**, f, i, h, **g**, **a**]	10	[c, **a**, f, i, **g**]
4	[d, a, b]	11	[i, **g**, **a**, b]
5	[e, **a**, **g**, f, i, j]	12	[f, i, j]
6	[i, h, g, f]	13	[**a**, **g**, f, i]
7	[h, i, **g**, **a**, b]	14	[d, a, b]

possible movement rule. This process is time consuming and memory intensive which is why efficient implementations have been studied in the literature.

An efficient implementation for sequence association rules generation is by performing vertical projection of the sequences and presenting *Unit Movement Pattern* with one *Sequence-Pattern Identity List* that shows where this item has shown up. Using this approach:

- The database is queried only once, and each unit movement's presence in sequences is projected into one list called *Trajectory Identity List* (TIL).
- To explore the support of combinations, it is enough then to query these lists.

Table 5
Example of Trajectory Identity Lists (TILs)

Unit Movement Pattern	Trajectory Identity List
a	[1,2,3,4,5,7,8,9,10,11,13,14]
b	[2,4,7,8,11,14]
c	[3,10]
d	[4,14]
e	[2,5,9]
f	[1,3,5,6,9,10,12,13]
g	[1,3,5,6,7,8,9,10,11,13]

For example, consider the *TrDB* shown in Table 4, the resulting *TILs* are shown in Table 5.

In order to calculate the support of a movement pattern it is enough to perform the intersection between the different unit movements' *TILs*.

Consider the case of calculating the support of $\{a, g\}$ using the explained algorithm. The support would then be the length of the list resulting from the intersection of a's and g's TILs. The resulting list would be [1,3,5,7,8,9,10,11,13] which is of length 9. This length should indicate the support of the movement pattern $\{a, g\}$ which is the number of its occurrence in the database. However, in the database, only 3 occurrences can be observed (highlighted in green) of this movement pattern. This error occurred because this approach fails to take into considerations the two following factors:

- the **order** of the items as well as
- the possibility of **recurrence** of the item in the same sequence.

Consequently, while this approach increases efficiency by decreasing the amount of database queries, applying this to vehicle trajectory database while these two critical factors are overlooked makes it produce wrong results.

3.3. New Definitions

In this study approach, it was necessary to add the following definitions and concepts:

1. Seeing as the sequences are trajectory movements, each edge can only be followed by a specific set of edges called set of outgoing edges and defined as follows:

$$O_x = \{y \in E \mid y \text{ is an outgoing edge from } x\} \tag{1}$$

Here, E represents the set of all edges in the graph, x represents an arbitrary edge in the set E, and y represents all edges that are outgoing from x. i.e. $O_a = \{b, f, g\}$ according to the graph in Fig. 1.
This property decreases substantially the search space, time, and complexity while generating the frequent sequences; instead of exploring all possible combinations, the set of outgoing edges is used to generate the next valid movement patterns.
2. *Positioning Table* (PT) is proposed in this study in order to project the order as well as the possible recurrence of a unit movement in a trajectory sequence. A PT is a two-dimensional table generated for a movement pattern that records all the positions where this pattern appears in the trajectory database (TrDB). Each row in this table records the positions of appearance of the movement pattern within a specific trajectory sequence using an *ordered-positioning list* (OPL) where each item in this list is a position of occurrence of this pattern within the specified sequence.

   - $OPL(M)_i$ represents the Ordered-Positioning List for movement pattern M in the trajectory sequence Tr_i. This list contains all the positions within this sequence where pattern M appears, recorded in a specific order. Mathematically, $OPL(M)_i$ can be represented as follows:

   $$OPL(M)_i = \{pos_j \mid pos_j \text{ is the position of } j^{th} \text{ occurrence of pattern M in } Tr_i, \text{ for } 1 \leq j \leq n_i\} \tag{2}$$

   In this equation n_i is the total number of occurrences of pattern M in the trajectory sequence Tr_i.
   - $PT(M)$ represents the Positioning Table for movement pattern M in the trajectory database TrDB. It contains only the Ordered-Positioning Lists (OPL_i) for pattern M where M exists in the i^{th} trajectory sequence. Mathematically, $PT(M)$ can be represented as follows:

   $$PT(M) = \{OPL(M)_i \mid M \text{ exists in the } i^{th} \text{ trajectory sequence}\} \tag{3}$$

   To summarize, $OPL(M)_i$ contains all positions of occurrence of pattern M in a sequence Tr_i, and $PT(M)$ contains the Ordered-Positioning Lists for pattern M in the entire trajectory database i.e. consider the unit movement patterns $\{a\}$ and $\{g\}$ and the $TrDB$ in Table 4, the generated positioning tables $PT(\{a\})$ and $PT(\{g\})$ are shown in Figs. 2a and 2b respectively.

3.4. Position Tables Extension Approach (PT-Ext)

To find the position of sequences the positioning tables are extended using the proposed strategy *PT-Ext*. Consider $PT(X)$ and $PT(Y)$ are the positioning tables of the movement patterns X and Y respectively. Y is a unit movement $\{y\}$, of order 1, and let X be of order k; $X = \{x_1, ..., x_k\}$. Considering that the following rules apply:

- Both X and Y belong to the frequent movement pattern sets FqM_k and FqM_1 respectively.
- y is an outgoing edge for the edge x_k ($y \in O_{x_i}$) where x_k is the k^{th} (last) unit movement in X.

Checking that all these rules apply, *PT-Ext* then goes over each row (Sequence id, i) in $PT(X)$ and checks for the following:

1. Index of X in S_i is not the last one otherwise it can't be extended.
2. The sequence id, i, exists in the tables of both movement patterns X and Y.
3. The index of y is directly subsequent to the index of the initial movement pattern X in S_i i.e. index of X in S_i is d and the index of y is $d + 1$.

Sequence (i)	OPL_i
1	(**4**)
~~2~~	(2)
3	(2, **8**)
~~4~~	(2)
5	(2)
7	(**4**)
8	(**2**)
9	(3)
10	(2)
11	(**3**)
13	(1)
~~14~~	(2)

(a) Positioning Table of $\{a\}$

Sequence (i)	OPL_i
1	(3)
3	(3, 7)
5	(3)
~~6~~	(3)
7	3)
8	(1)
9	(7)
10	(5)
11	(2)
13	(2)

(b) Positioning Table of $\{g\}$

Sequence (i)	OPL_i
3	(**3**)
5	(**3**)
13	(**2**)

(c) Positioning Table of $\{a, \mathbf{g}\}$

Sequence (i)	OPL_i
1	(**4**)
3	(**8**)
7	(**4**)
8	(**2**)
11	(**3**)

(d) Positioning Table of $\{g, \mathbf{a}\}$

Figure 2: Example joining positioning tables

If both rules apply, the index of the subsequent movement pattern is added to the movement pattern positioning list for the new extended sequence, $OPL(X')$, where X' is the extension of X by y aka $X' = \{x_1, ..., x_k, y\}$.

In the tables shown in 2a and 2b, the initial positioning tables of $\{a\}$ and $\{g\}$ respectively are shown. Extending the positioning table of $\{a\}$ by $\{g\}$ since g is an outgoing edge for a, the sequence ids that show up in both tables $PT(\{a\})$ and $PT(\{g\})$ have to be checked (discarding sequences 2, 4 and 6). Then, the remaining common sequences' order-positioning lists are checked to see whether the index of the considered outgoing edge $index(\{g\})$ is $index(\{a\}) + 1$. As shown in the tables, that is the case in the sequences 3, 5, and 13. Now in the positioning table of $\{a, g\}$, for these three sequences, a list is added that contains the index of $\{g\}$, the subsequent pattern. The resulting PT of $\{a, g\}$ is shown in Fig. 2c.

Extending position tables using *PT-Ext* makes sure that all the errors previously discussed are considered and solved. Furthermore this algorithm is more efficient as it only extends frequent patterns, excluding non-frequent patterns, and explores only possible extensions, those included in the set of outgoing edges of the last unit movement as opposed to exploring all (frequent) edge combinations.

3.5. VeTraSPM Model

This paper presents a new approach for trajectory sequences pattern mining abbreviated *VeTraSPM*. VeTraSPM employs a path *Positioning Table* abbreviated PT for data storing, mining and pattern expansion. The PT of each single-edge-path are firstly obtained through scanning sequence database once. Then PTs are mined to identify the frequent single items. Based on the new expansion strategy named *PT-Ext*, the frequent patterns are extended and the extended tables are created and updated. The extended PTs then are explored to obtain the frequent extended patterns. Finally, a pruning technique is used for to avoid the generation of unnecessarily large number of candidate patterns.

Figure 3 illustrates the sequential flow of operations in the VeTraSPM algorithm, designed to identify frequent movement patterns in vehicle trajectory data. The process consists of several key steps:

1. **Generate Outgoing Edges:** The algorithm begins by analyzing the map to identify all *outgoing edges* for each edge in the network. This step captures the possible connections between road segments based on their physical connectivity, such as junctions or intersections.
2. **Construct Positioning Tables:** Next, the vehicle trajectory database is scanned to *construct positioning tables* for movement patterns of order k. These tables map the locations of movement sequences across the dataset, helping to keep track of where specific road sequences occur.
3. **Prune to Obtain Frequent Patterns (FqM_k):** The positioning tables are then pruned to extract the frequent movement patterns FqM_k of order k. This step ensures that only movement patterns meeting a minimum frequency threshold are retained.
4. **Check if FqM_k is Empty:** After generating the frequent movement patterns for the current order k, the algorithm checks if the set FqM_k is empty.
    - *If FqM_k is not empty*, the algorithm proceeds to extend the patterns to the next order, $k + 1$.

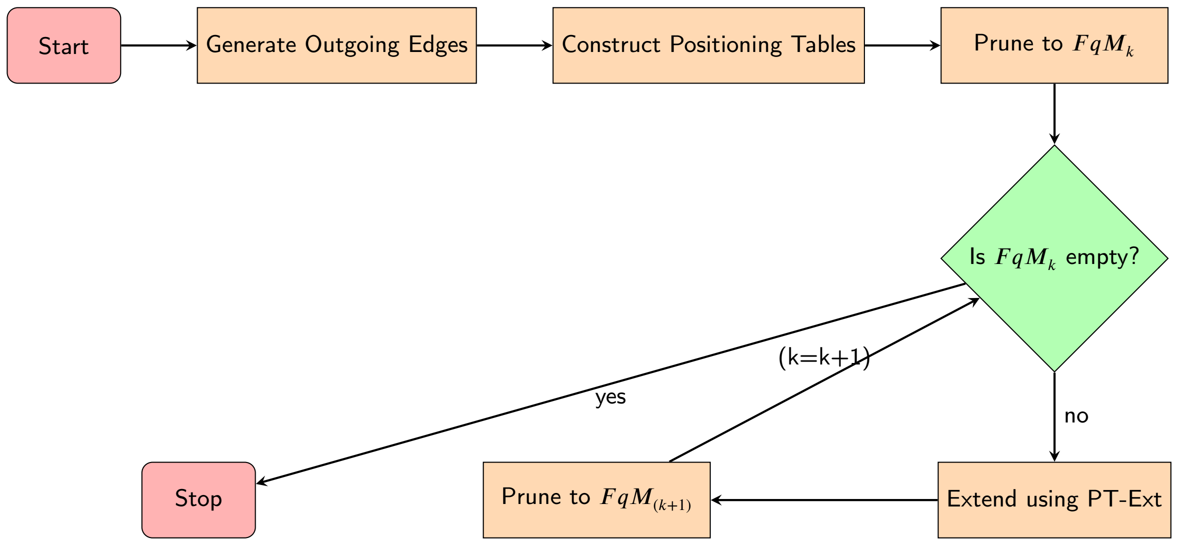

Figure 3: VeTraSPM

- *If FqM_k is empty*, the algorithm terminates, as no further patterns can be generated.

5. **Extend Patterns using PT-Ext:** When the frequent patterns FqM_k are non-empty, they are *extended to higher-order patterns* $(k+1)$ using the *PT-Ext joining strategy*. PT-Ext ensures efficient extension by merging compatible patterns and generating new positioning tables for the extended patterns.
6. **Recursive Pruning for Higher-Order Patterns:** The newly generated positioning tables for higher-order patterns are pruned again to obtain the next set of frequent movement patterns, $FqM_{(k+1)}$. This process ensures that only significant patterns are retained at each step.
7. **Termination Condition:** The process repeats recursively, generating and extending frequent patterns until no new patterns can be found, i.e., when the set FqM_k becomes empty. At this point, the algorithm stops.

3.6. Sequential Impact Scores (SIS): New Criticality Indices based on VeTraSPM

In the context of this study, let n represent the order of the last non-empty set of frequent movement patterns. Consider a specific movement pattern M, to gauge the significance of this pattern across various orders of frequent movement patterns, the following novel metric is introduced: SIS.

Let $FqMS(M)$ be the frequent movement pattern score of movement pattern M and $CMS(S)$ be the confident pattern score of pattern M.

$$FqMS(M) = 1 \cdot x_1 + \left(\frac{1}{i}\right) \cdot x_i + \ldots + \left(\frac{1}{n}\right) \cdot x_n \tag{4}$$

$$CMS(M) = 1 \cdot y_1 + \left(\frac{1}{i}\right) \cdot y_i + \ldots + \left(\frac{1}{n}\right) \cdot x_n \tag{5}$$

Here's the breakdown of the equations:

- The terms $1 \cdot x_1$ and $1 \cdot y_1$ signify the contribution of occurrences of pattern M in the first-order set of frequent movement patterns (FqM_1) and the first-order set of confident movement patterns (CM_1) respectively.
- The terms $\left(\frac{1}{i}\right) \cdot x_i$ and $\left(\frac{1}{i}\right) \cdot y_i$ encapsulates the importance of pattern M in the i^{th} order set of frequent movement patterns (FqM_i) and confident movement patterns (CM_i) respectively. Here, the factor $\frac{1}{i}$ is utilized to weigh patterns by their order inversely, assigning higher importance to lower-order patterns.

- Similarly, the terms $\left(\frac{1}{n}\right) \cdot x_n$ and $\left(\frac{1}{n}\right) \cdot y_n$ take into account the occurrences of pattern M in the last non-empty set of frequent movement patterns (FqM_n) and confident movement patterns (CM_n), with a weight proportional to $\frac{1}{n}$.

The Sequential Impact Score of movement pattern M is defined by the following equation:

$$SIS(M) = FqMS(M) + CMS(M) \tag{6}$$

In essence, the Sequential Impact Score $SIS(M)$ combines the occurrences of pattern M across different orders, where the weight assigned to each order is inversely proportional to its value. This makes sure that the most frequent and confident patterns have been captured and that links were assigned higher importance according to their consistent appearance in the frequent movement patterns and confident movement patterns of higher order. These indices provide a comprehensive assessment of the sustained significance of a movement pattern across diverse levels of sequence complexity.

By calculating the Sequential Impact Scores for each movement pattern, insights can be gained into the patterns that consistently exhibit significance across different orders of frequent movement patterns and confident movement patterns. This information proves invaluable for decision-making and comprehending how the relevance of patterns evolves.

4. Implementation of VeTraSPM and SIS Calculation

This section describes the implementation of VeTraSPM and SIS calculation, along with optimizations aimed at improving memory efficiency, scalability, and computational performance. Several strategies are employed, including lazy evaluation, partitioning, memory-efficient data structures, early termination, and parallelization, to ensure the algorithm can handle large datasets efficiently. A comparative analysis of the original and optimized versions is provided to demonstrate the benefits.

4.1. Research Flow

The research follows a structured flow consisting of the following major steps, shown in Figure 4.

- Data Preparation: The `DataPreparation` function takes an `xml_file` as input and processes it to extract the necessary data. It first imports the data using `ImportData` and then converts this data into a directed graph using `CreateDirectedGraph`. The function returns the generated graph for further processing.

- Edge & Table Construction: The second function, `EdgeAndTableConstruction`, receives the directed graph as input. This function is responsible for generating outgoing edges from the graph using `GenerateOutgoingEdges`. Additionally, it constructs positioning tables from the outgoing edges using `CreatePositioningTables`. The resulting positioning tables are returned to continue the mining process.

- Frequent Pattern Mining: In the `FrequentPatternMining` function, the positioning tables are used to extract frequent patterns. Initially, the patterns are pruned based on a support threshold using `PatternPruning`. After that, the pruned patterns are extended using `ExtendPatterns`. A loop is executed to iteratively prune and extend the patterns until no new frequent patterns are found. This ensures that only the most relevant patterns are retained. The frequent patterns are then returned for final analysis.

- Final Analysis: `FinalAnalysis` processes the frequent patterns to calculate SIS using the `CalculateSIS` function. Once SIS is calculated, critical links are identified in the network through the `IdentifyCriticalLinks` function.

- Visualization: The final step is carried out where the identified critical links are visualized for interpretation using `VisualizeCriticalLinks`.

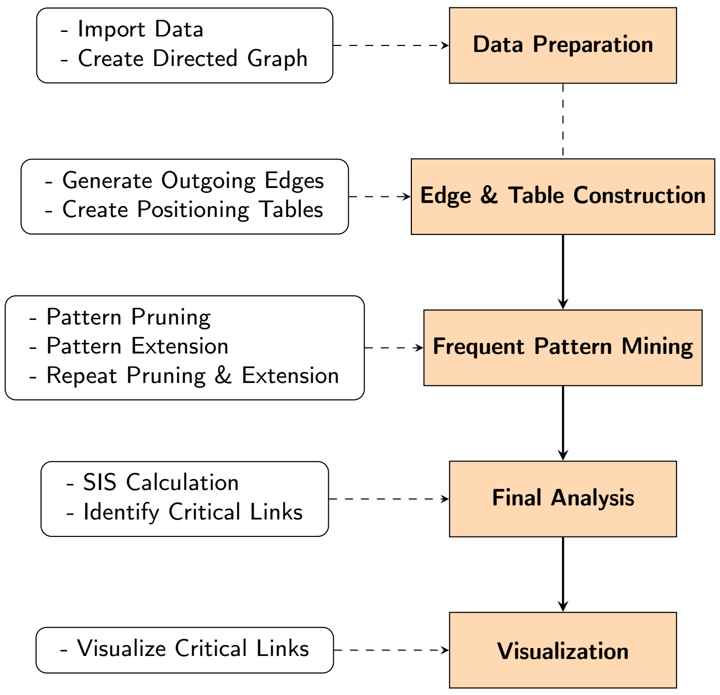
- Import Data
- Create Directed Graph
Data Preparation
- Generate Outgoing Edges
- Create Positioning Tables
Edge & Table Construction
- Pattern Pruning
- Pattern Extension
- Repeat Pruning & Extension
Frequent Pattern Mining
- SIS Calculation
- Identify Critical Links
Final Analysis
- Visualize Critical Links
Visualization

Figure 4: Model Flow

4.2. Generate Outgoing Edges and Positioning Tables Construction

The first step is to read the simulation network as a traffic road network model. In this work, a directed multi-graph is constructed from the city map using the `networkx` library, maintaining the traffic network's connections and circulation patterns. A directed multi-graph supports multi-directional edges and loops, allowing a realistic representation of road networks. Each road is mapped as an edge-weighted by length, cost, and average speed.

Once the graph is built, the outgoing edges for each road link are extracted. This is achieved through Algorithm 1, which iterates over the graph, mapping each edge to its corresponding outgoing edges. A directed multi-graph representation of the traffic network is loaded, and for each edge in the graph, the outgoing edges are identified. The algorithm iterates over all edges and stores their outgoing edges in a dictionary. The outgoing edges are stored as NumPy arrays for more efficient indexing and numerical operations. The time complexity is $O(E)$, where E represents the total number of edges. With partitioning, the workload is distributed into smaller subsets, reducing the complexity to $O(E/P)$, where P is the number of partitions.

For the second step, the vehicle driving trajectories database (TrDB) is first loaded. By processing the data only when necessary, lazy evaluation is employed, thus reducing peak memory usage. Algorithm 2 shows how instead of loading the entire XML file and its contents into memory all at once, Vehicle routes can be processed one at a time and yield the result incrementally. This way, the data is processed on demand rather than all at once greatly reducing the space complexity from $O(V \cdot M)$ where V is the number of vehicles and M is the average number of movements per vehicle to $O(1)$. This strategy is especially useful when dealing with large datasets, as it minimizes memory usage while maintaining the same time complexity.

TrDB is then scanned once to construct the initial unit movements positioning tables which is the vertical projection of the database. These tables indicate for each edge in which routes it has appeared and at which position. Algorithm 3 takes the vehicle driving trajectories database (TrDB) named `tr_db` as input and produces the dictionary `edge_pt`. The algorithm begins by initializing an empty dictionary called `edge_pt`. It then lazily iterates through each trajectory in `tr_db`, which is processed one by one to avoid loading the entire dataset into memory. For each trajectory (route), the algorithm iterates through the enumerated list of unit movements (edges). During this iteration, the index of each unit movement within the trajectory is appended to the list associated with the corresponding unit movement and trajectory ID in the dictionary `edge_pt`. The algorithm returns the completed dictionary `edge_pt`, where each unit movement is mapped to its respective positioning table as a Numpy array, containing the trajectory IDs and

Algorithm 1 Construct Edge Outgoing Dictionary

```
Require: G (directed multi-graph representation of the map), P (number of partitions)
Ensure: out_edges (dictionary mapping each edge to its corresponding array of outgoing edges)
 1: Initialize an empty dictionary called edge_out_dict
 2: Partition G into P subgraphs G1, G2, ..., GP for parallel processing
 3: for each partition Gp in parallel do
 4:     for edge in Gp do
 5:         out_edges[edge] = np.array(edge.out_edges())   ▷ Store as a NumPy array for efficient access
 6:     end for
 7: end for
 8: return out_edges
```

Algorithm 2 Parse TrDB from XML File

```
Require: xml_path (path to an XML file containing vehicle routes)
Ensure: A generator yielding each vehicle's route as a list of movement units
 1: for veh_route in xml_path do
 2:     yield veh_route.edges()                    ▷ Yield each vehicle route instead of appending
 3: end for
```

Algorithm 3 Construct Unit Movement (Edge) Positioning Tables

```
Require: tr_db (generator yielding vehicle driving trajectories (TrDB)), P (number of partitions)
Ensure: edge_pt (dictionary mapping each unit movement to a NumPy array for its positioning table)
 1: Initialize an empty dictionary called edge_pt
 2: Split tr_db into P partitions
 3: for partition in tr_db do                                      ▷ Process partitions in parallel
 4:     for traj in partition do
 5:         for um_index, unit_movement in enumerate(traj) do
 6:             if unit_movement not in edge_pt then
 7:                 edge_pt[unit_movement] = np.empty((0,), dtype=int)    ▷ Initialize NumPy array
 8:             end if
 9:             edge_pt[unit_movement] = np.append(edge_pt[unit_movement], um_index)
10:         end for
11:     end for
12: end for
13: return edge_pt
```

indices for efficient sequential pattern mining. Employing partitioning and parallelization as well minimizes memory consumption, reducing the time complexity from $O(V \cdot M)$ to $O((V \cdot M)/P)$.

Using this positioning table concept, the space and time complexity are greatly reduced compared to parsing the original table each time. Parsing the original table at each order would be space and computationally expensive as it would be of the time complexity $O(N \cdot V \cdot M)$ where N is the total number of movement patterns at all levels. In contrast, the time complexity of parsing the partitioning table at each order k would be $O(N_{(k-1)} \cdot M_{(k-1)})$ where N_i is the number of movement patterns at order i only and M_i is the average number of occurrences of the movement patterns of order i. The space complexity is similarly reduced as only the order-specific movement patterns are considered and their specific indices.

4.3. Pruning and Extending Positioning Tables

To identify frequent movement patterns, low-frequency patterns are pruned using Algorithm 4. This algorithm calculates the support of each pattern and retains only those that meet the minimum support threshold. The given Algorithm 4 takes the table `init_pt`, table of positioning tables of current order k movement patterns, and a minimum support value `min_sup` as input.

Algorithm 4 Prune Movement Patterns by Minimum Support

```
Require: init_pt (dict), min_sup, P (number of partitions)
Ensure: frequent_movement_patterns_list (list)
 1: Split init_pt into P partitions
 2: Initialize an empty list called frequent_movement_patterns_list
 3: for partition in init_pt do                                ▷ Process partitions in parallel
 4:     for mv_pattern in partition do
 5:         total_support ← 0
 6:         is_frequent ← False
 7:         for traj_id in init_pt[mv_pattern] do
 8:             total_support ← total_support + len(init_pt[mv_pattern][traj_id])
 9:             if total_support ≥ min_sup then
10:                 Add mv_pattern to frequent_movement_patterns_list
11:                 is_frequent ← True
12:                 break
13:             end if
14:         end for
15:         if is_frequent then
16:             break
17:         end if
18:     end for
19: end for
20: return frequent_movement_patterns_list
```

It iterates through each movement pattern in the table, calculating the total support for each by summing the lengths of its list of positions in the trajectories associated with it. Once the total support of a movement pattern exceeds or equals the specified min_sup, it is returned in a list called frequent_movement_patterns_list. Finally, the algorithm returns the resulting list (FqM_k), which contains movement patterns of order k that meet the minimum support threshold. Before proceeding to the next step, the generated list (FqM_k) is saved in the database for later use. Also, the corresponding confident movement pattern set (CM_k) of the same order is calculated using $PT(FqM_k)$ and $PT(FqM_1)$ and saved. Considering the exhaustive case where each movement pattern appears no more than one time in a trajectory, the time complexity of Algorithm 4 at each level k would be $O(N_k \cdot M_k)$. Using partitioning, this complexity is improved to $O(N_k \cdot M_k / P)$.

The identified frequent patterns are extended using the PT-Ext algorithm (Algorithm 5). This process involves adding valid extensions to existing patterns, based on the outgoing edges of the last road link in each pattern. The algorithm takes as input:

- the resulting list of frequent movement patterns of order k, frequent_movement_patterns_list,
- the initial positioning table init_pt of movement patterns of order k; first time around it's the same as edge_pt (k=1),
- the edge positioning table edge_pt of unit movement patterns of order 1,
- last_edge_table, a table containing for each unit movement pattern (edge) a list indicating the ids of the trajectory sequences it is the last edge in (final destination),
- the dictionary of **frequent** outgoing edges for each unit movement pattern frequent_out_edges which resulted from pruning out_edges and edge_pt by minimum support.

The result is the new "extended" positioning tables of movement patterns of order $k + 1$, pt_extended.

The idea in this algorithm is to go over only the already identified frequent movement patterns of order k and extend them with only the identified frequent outgoing edges of their last unit movement pattern. The algorithm initializes an empty dictionary called pt_extended. It then starts by iterating over frequent movement

Algorithm 5 Positioning Table Extension (PT-Ext)

```
Require: frequent_movement_patterns_list, init_pt, edge_pt, last_edge_table,
    frequent_out_edges, P (number of partitions)
Ensure: pt_extended
 1: Initialize an empty dictionary called pt_extended
 2: Split frequent_movement_patterns_list into P partitions
 3: for partition in frequent_movement_patterns_list do                ▷ Process in parallel
 4:     for movement_pattern in partition do
 5:         mp_pt ← init_pt[movement_pattern]
 6:         last_edge ← movement_pattern[-1]
 7:         for index in mp_pt[trajectory_id] do
 8:             if trajectory_id not in last_edge_table[last_edge] then
 9:                 for out_edge in frequent_out_edges[last_edge] do
10:                     if trajectory_id in edge_pt[out_edge] then
11:                         for out_index in edge_pt[out_edge][trajectory_id] do
12:                             if out_index = index + 1 then
13:                                 movement_id ← movement_pattern ⊕ out_edge
14:                                 pt_extended[movement_id][trajectory_id].append(out_index)
15:                             end if
16:                         end for
17:                     end if
18:                 end for
19:             end if
20:         end for
21:     end for
22: end for
23: return pt_extended
```

patterns (`frequent_movement_patterns_list`). For each movement pattern (`movement_pattern`), it retrieves its positioning table (`mp_pt`). In order to increase efficiency, `last_edge_table` are used to exclude the trajectory ids where the last edge of the considered movement pattern is the last one in. After that, the algorithm goes over the frequent outgoing edges of the last edge (`last_edge`) in this movement pattern. For each frequent outgoing edge (`out_edge`), the algorithm iterates through the trajectory IDs and their corresponding indices in the `mp_pt`. If the trajectory ID is present in both its positioning table and that of the outgoing edge, and the outgoing edge index is one greater than the that of the considered movement pattern, a new movement pattern is constructed by extending the initial movement pattern with that outgoing edge. The algorithm adds the extension index to the `pt_extended` dictionary under the new movement pattern id and trajectory ID. Finally, the extended positioning table (`pt_extended`) is returned by the algorithm. This positioning table is now of movement patterns one order greater than the movement patterns in the initial positioning table which is now of order $k + 1$.

The proposed algorithm focuses on avoiding exploring all the different combinations and appending all different movement patterns since only the outgoing edges of the last edge in the movement pattern are viable concatenations at each point. This greatly reduces the search space as well as the time complexity of the extension process. Instead of exploring $O(N^n)$ where n is the max order reached with discover-able frequent sequences, $O(N \cdot E_{out})$ is explored where E_{out} is the average number of outgoing edges for each edge. Also, by checking whether the last edge in the movement pattern is not the last edge in the trajectory before proceeding, unnecessary iterations are avoided which further reduces time complexity. The algorithm's time complexity is $O(N_k \cdot E_{out} \cdot M_k)$. Considering that the average number of outgoing edges is negligible comparably, then the time complexity is basically $O(N_k \cdot M_k)$. By using partitioning and parallelization however it becomes $O((N_k \cdot M_k)/P)$.

4.4. VeTraSPM Implementation

VeTraSPM is implemented by looping over and over in a recursive process of pruning the generated higher order movement patterns and extending them until the pruning results in an empty list as shown in Algorithm 6. The resulting

Algorithm 6 VeTraSPM

```
Require: min_sup, edge_pt, out_edges, last_edge_table, P (number of partitions)
Ensure: fqmp_res, cmp_res
 1: Initialize an empty list fqmp_res
 2: Initialize an empty list cmp_res
 3: Initialize order_nb to 1
 4: frequent_movement_patterns_list ← prune_by_support(edge_pt, min_sup)
 5: if len(frequent_movement_patterns_list) < 1 then
 6:     return False                          ▷ Early termination if no frequent patterns
 7: else
 8:     fqmp_res.append(np.array(frequent_movement_patterns_list))
 9:     cmp_res.append(np.array([]))
10: end if
11: Initialize an empty dictionary frequent_out_edges
12: for x in partition do
13:     frequent_out_edges[x] ← [out_edge for out_edge in out_edges[x] if out_edge in
    frequent_movement_patterns_list]
14: end for
15: pt_extended ← edge_pt
16: Increment order_nb by 1
17: while True do
18:     pt_extended ← pt_ext(frequent_movement_patterns_list, pt_extended, edge_pt,
    last_edge_table, frequent_out_edges, P)
19:     frequent_movement_patterns_list ← prune_by_support(pt_extended, min_sup, P)
20:     if len(frequent_movement_patterns_list) < 1 then
21:         break                             ▷ Early termination to stop recursion
22:     end if
23:     Initialize an empty dictionary confident_rules
24:     edge_support ← get_sequences_support(edge_pt)
25:     sequences_support ← get_sequences_support(pt_extended)
26:     for movement_pattern in frequent_movement_patterns_list do
27:         edge_0 ← movement_pattern.split()[0]
28:         rule_conf ← sequences_support[movement_pattern] / edge_support[edge_0]
29:         if rule_conf ≥ 0.6 then
30:             confident_rules[movement_pattern] ← rule_conf
31:         end if
32:     end for
33:     fqmp_res.append(np.array(frequent_movement_patterns_list))
34:     cmp_res.append(np.array(confident_rules))
35:     Increment order_nb by 1
36: end while
37: return fqmp_res, cmp_res
```

PT tables from Algorithm 5 are passed into Algorithm 4 and the resulting set of frequent patterns $FqM_{(k+1)}$ is passed again into Algorithm 5, over and over until the resulting set of frequent patterns resulting from Algorithm 4 is empty.

Each recursion involves pruning $O(N_k \cdot M_k/P)$ and extending $O((N_k \cdot M_k)/P)$. Hence the time complexity of VeTraSPM is $O(n \cdot (N_k \cdot M_k)/P)$, where n is the number of recursive steps (orders). The time complexity increases with the number of orders n, but each pruning and extension step ensures that only relevant patterns are kept, reducing unnecessary computations.

4.5. Calculating Sequential Impact Score

For the purpose of calculating *SIS* of each edge, as mentioned before, after each pruning performed by Algorithm 4, the resulting frequent movement patterns and confident movement pattern lists of order k, FqM_k and CM_k are saved. When the whole process is done and the recursive process ends, the FqM and CM_k lists are generated and for each edge appearance they are parsed and its *SIS* value is updated according to the number of its occurrences in the lists at each order as shown in Algorithm 7.

The time complexity of SIS calculation is $O(n \cdot F \cdot E)$, where n is the number of iterations (orders), F is the number of frequent patterns, and E is the average number of edges per movement pattern. Using partitioning, the complexity reduces to $O((n \cdot F \cdot E)/P)$, making it suitable for large-scale trajectory datasets.

Algorithm 7 Calculate Sequential Impact Score

```
Require: fqmp_res, cmp_res, P (number of partitions)
Ensure: sis (NumPy array of SIS values)
 1: Initialize a NumPy array sis with zeros for each edge
 2: Split fqmp_res into P partitions
 3: Split cmp_res into P partitions
 4: for partition in fqmp_res do                                ▷ Process in parallel
 5:     for order_nb, order in enumerate(partition) do
 6:         for fmp in order do
 7:             for edge in fmp do
 8:                 if edge ∉sis then
 9:                     sis[edge] ← $\frac{1}{\texttt{order\_nb + 1}}$
10:                 else
11:                     sis[edge] ← sis[edge] + $\frac{1}{\texttt{order\_nb + 1}}$      ▷ Efficient NumPy-based update
12:                 end if
13:             end for
14:         end for
15:     end for
16: end for
17: for partition in cmp_res do                                 ▷ Process in parallel
18:     for order_nb, order in enumerate(partition) do
19:         for cmp in order do
20:             for edge in cmp do
21:                 sis[edge] ← sis[edge] + $\frac{1}{\texttt{order\_nb + 1}}$
22:             end for
23:         end for
24:     end for
25: end for
26: return sis
```

4.6. Optimization Strategies Used

As mentioned earlier, several optimization strategies were employed to improve the performance of the algorithm. By applying these strategies, the **VeTraSPM algorithm becomes scalable** to larger datasets. The recursive processes are sped up by partitioning, parallelization, and early termination. Memory-efficient structures and lazy evaluation reduce peak memory usage, improving scalability for large datasets as well. The improvements in both **time and space complexity** help ensure that the algorithm can handle real-world trajectory data efficiently without running into **memory bottlenecks**.

5. Experimentation and Evaluation

This section presents the experimentation and evaluation of this study. The proposed algorithm is compared with the efficient implementations of Apriori algorithm used in works [2].

5.1. Experimental set-up and data sources

The experiments were conducted on a machine running **Windows 11** with **32GB RAM** and an **AMD Ryzen 9 6900HS @3.30GHz** (8 cores, 16 threads). All cores were utilized for parallel processing tasks, such as partitioning and processing movement patterns, edge dictionary construction, and positioning table extensions. *NumPy arrays* and *lazy evaluation* via generators were used to manage memory efficiently. The peak memory usage remained within the available 32GB RAM, while the CPU parallelization reduced execution time. No GPU acceleration was used, and data was streamed into memory to prevent overwhelming the system during XML parsing.

The chosen micro-simulation tool for the critical link analysis phase is the SUMO simulator. In this study, existing realistic scenario LuST [1] of the city of Luxembourg is used. Additionally, the MoST scenario [30] based on Monaco is also used in this work to validate the framework's applicability across cities of different sizes and transportation characteristics. The details and statistics for both scenarios are outlined in Table 6.

Table 6
SUMO Simulation Scenario Numbers

Ref.	Year	Name	City	Area	Total Nodes	Total Edges	Total Trips
[1]	2017	LuSTScenario	Luxembourg	155.95 km2	2,247	5,779	215,526
[30]	2018	MoSTScenario	Monaco	22 km2	2,004	4,404	7,990

As explained in the work [1], to achieve authentic traffic patterns in the SUMO LuST (Luxembourg scenario), the simulation relies on genuine demographic data provided by the government. For the city's public transport component, data from the public transport database is utilized to obtain information about bus routes. The overall traffic demand encompasses buses, local, and transit mobility. While buses and transit mobility follow predetermined routes, local mobility requires generation and optimization to simulate realistic traces.

To assess the realism of the LuST Scenario's traffic demand, a dataset collected in the City of Luxembourg between March and April 2015 was used by the authors. This dataset comprises over six million Floating Car Data (FCD) samples from more than 14 thousand trips conducted between 06:00 and 22:00. Comparisons revealed similar distributions between the simulated scenario and the real dataset, indicating that LuST is capable of providing realistic traffic demand and mobility traces.

The evaluation further demonstrated that speed distributions in the simulation closely resemble those in the real dataset, with discrepancies primarily attributed to the absence of pedestrian mobility rather than changes in the road topology. Additionally, their results highlighted that dynamic rerouting enhances the interactive scenario's realism, closely aligning with precomputed optimized mobility patterns.

5.2. Parameters, Values, and Justification

In this study, the frequency of pattern occurrences serves as a quantitative measure of the prevalence of specific movement patterns within the dataset. Counting the occurrences of patterns provides us with a foundation for prioritizing patterns for further analysis. By focusing on patterns with higher occurrence frequencies, the understanding of dominant vehicular behaviors is refined while maintaining a data-driven perspective.

In order to do this, two key parameters have been identified, each with specific values and justifications. These parameters play a crucial role in fine-tuning this approach to extract meaningful patterns and relationships from vehicle trajectory data.

5.2.1. Minimum Support Threshold

The minimum support threshold serves as a fundamental parameter in this methodology. It determines the threshold frequency a pattern must satisfy to be considered for further analysis. To comprehensively explore patterns across various popularity orders, a spectrum of threshold values was chosen: 3%, 5%, 8%, 10%, and 12% whose respective minimum number of occurrences is (6,465), (10,776), (17,242), (21,552), and (25,863) respectively. This range enables us to strike a balance between capturing rare patterns that might offer unique insights and identifying frequently occurring patterns that could indicate important trends.

As the minimum support threshold is manipulated, a notable phenomenon arises: the frequency of identified patterns changes. As shown in Figure 5, higher thresholds lead to a decrease in the number of identified patterns, as

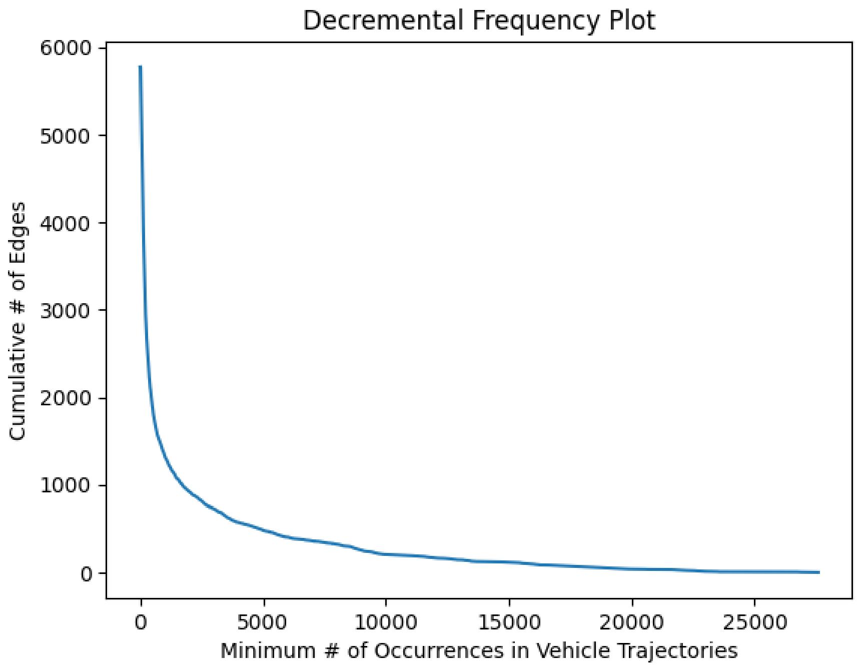

Figure 5: Cumulative Nb. of Frequent Edges by Minimum Nb. of Occurrences Threshold

patterns must surpass a higher popularity bar to be considered. This relationship is critical as it ensures that the method remains sensitive to the minimum support threshold while carefully curating patterns with substantive implications.

5.2.2. Minimum Confidence Threshold

The confidence threshold parameter is central also for the criticality link analysis process. It quantifies the strength of the rules derived from patterns, indicating the reliability of the associations between different trajectory events. The confidence threshold is expressed as a variable fraction, allowing us to adapt its value according to the characteristics of the dataset under investigation. This adaptability ensures that the rules generated accurately reflect the inherent uncertainty present in real-world vehicle movement data. In this work, different confidence values were explored: 0.6, 0.7, 0.8, 0.9, and 1. As this value increases only edges in more “confident” rules and corresponding patterns are assigned higher weight.

5.2.3. Choosing Minimum Support and Confidence Thresholds

The delicate balance between including less common patterns and excluding exceedingly frequent ones is a nuanced consideration in the proposed methodology. Striking this balance ensures that both the long-tail patterns that might provide unique insights and the highly frequent patterns that may underscore critical vehicular interactions are captured. Studying different minimum support thresholds, a variation in the max order reached was observed. As the threshold increases, the max order reached decreases as shown in Figure 6.

Since the confidence threshold is also central in this algorithm, the variation of the confident rule (pattern) counts generated as different levels across different minimum support and confidence thresholds were studied. In order to visualize this variation, a heatmap is used as shown in Figure 7.

In order to strike a balance in a way so that the proposed algorithm remains versatile and adaptable to the analytical objective, a minimum support has to be chosen that is able to reach higher levels but also capture only more frequent movement patterns. Similarly, a minimum confidence threshold has to be chosen that according to the chosen support generates enough confident rules (patterns) to satisfy the algorithm at different orders.

For this reason in the experimentation 3% and 0.8 were used as the minimum support and confidence thresholds respectively. These minimum support and confidence thresholds ensure that a sufficient amount of frequent movement patterns are explored and a sufficient confident movement patterns are generated.

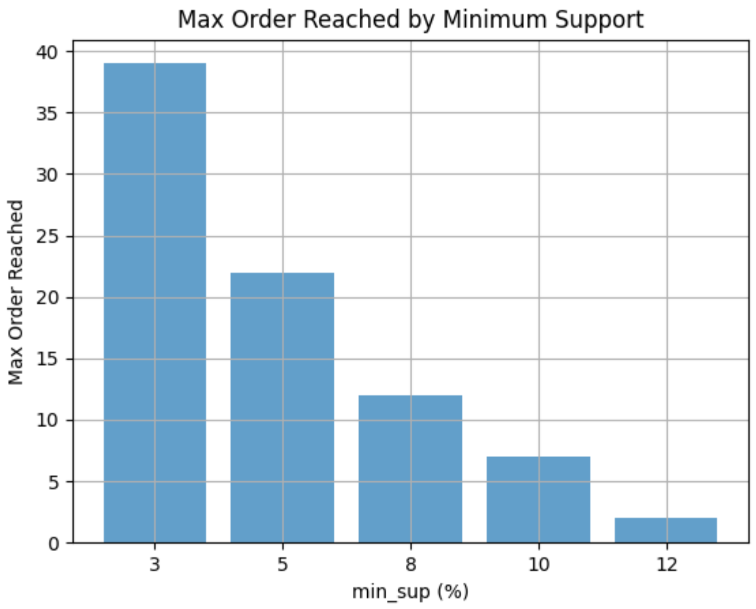

Figure 6: Maximum Order Reached by Minimum Support

5.3. Results

To present the outcomes of this study, the results are divided into two key parts. The first part is dedicated to the efficiency and accuracy evaluation of VeTraSPM, where its performance is assessed against standard Apriori, enhanced Apriori, PrefixSpan, and SPADE. The second part delves into the results of the proposed SIS, providing insights into the network's critical links based on both frequent and confident movement patterns. Together, these results offer a comprehensive understanding of the algorithmic efficiency and impact assessment capabilities of the proposed methodologies.

5.4. Execution Time Results

The execution time of VeTraSPM is compared with several baseline algorithms: base Apriori, enhanced Apriori (which explores outgoing edges first), PrefixSpan, and SPADE. The comparison was made across varying minimum support (min_sup) thresholds, from 3% to 12% as shown in Figure 8. At lower thresholds like **3% min_sup**, VeTraSPM completes the task much faster than the other algorithms, where base Apriori and enhanced Apriori take significantly longer. As the **min_sup** increases, VeTraSPM continues to outperform, taking only a fraction of the time required by the other methods. For example, at **5% min_sup**, it remains significantly faster, with enhanced Apriori, base Apriori, and the other methods requiring much longer execution times. At higher thresholds, such as **10%** and **12%**, the execution times across all algorithms converge further, but VeTraSPM still maintains its edge, completing much faster, even when the search space reduces. Overall, Figure 8 demonstrates how VeTraSPM consistently outperforms the baseline methods across all thresholds, particularly at lower support levels where computational demand is higher.

VeTraSPM shows substantial efficiency gains, outperforming base Apriori by an order of magnitude at lower min_sup values due to reduced candidate generation. Despite enhancements, the Apriori variant lags behind VeTraSPM, which benefits from its optimized memory handling and vertical projection. VeTraSPM outperforms both PrefixSpan and SPADE at lower min_sup values, handling the sequential and repetitive nature of trajectory data more effectively. While PrefixSpan and SPADE improve with higher min_sup, VeTraSPM remains consistently faster.

VeTraSPM remains efficient across varying min_sup values, handling large datasets effectively. Its time complexity is reduced through vertical projection, and it uses memory more efficiently, avoiding the computational overhead seen in base Apriori.

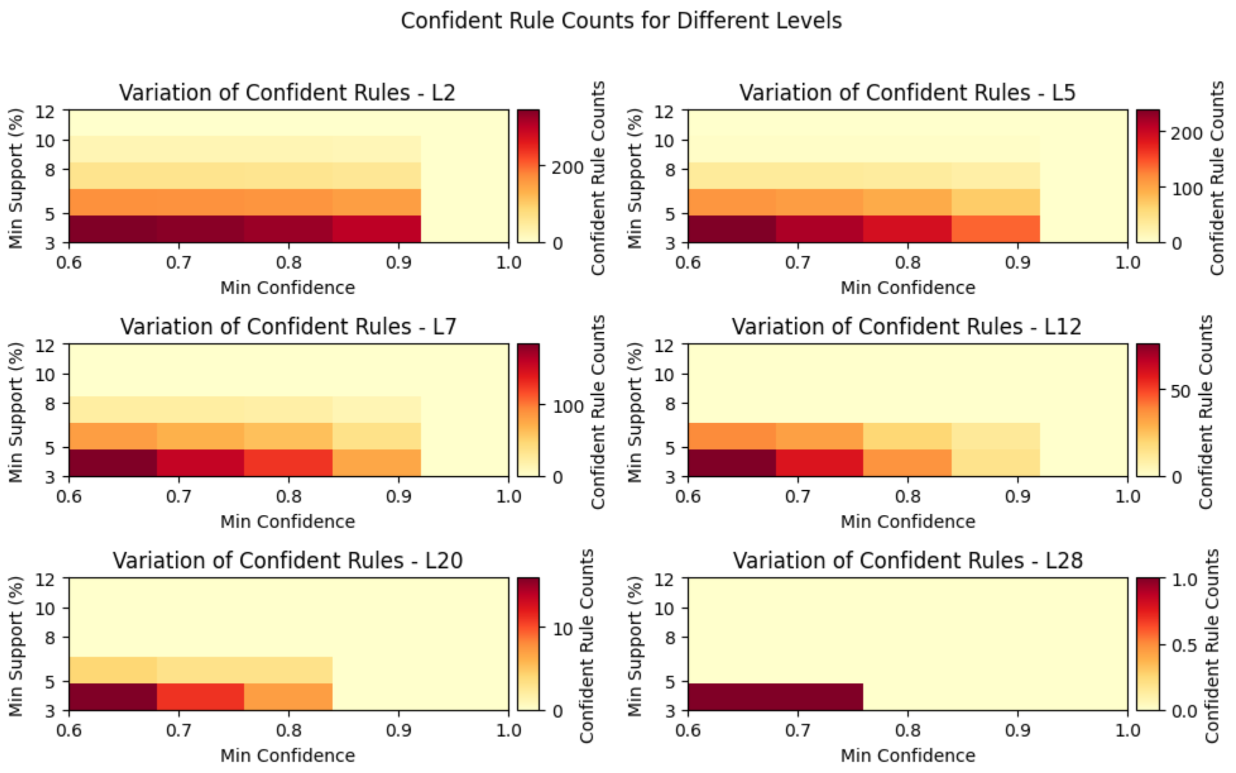

Figure 7: Confident Rule Counts for Different Orders

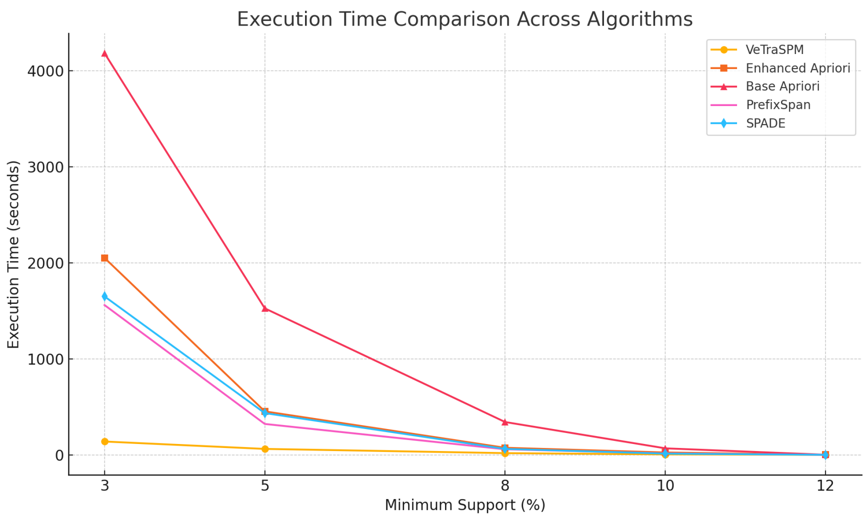

Figure 8: VeTraSPM Execution Time vs. Existing Implementation

5.4.1. SIS Results

In this study, both the sets of frequent movement patterns (FqM) and confident movement patterns (CM) are leveraged to compute the proposed SIS.

When visualizing the impact score derived from FqM, illustrated in Figure 9, the focus in this study is directed towards the most commonly traversed edges and sequences, placing particular emphasis on edges embedded within

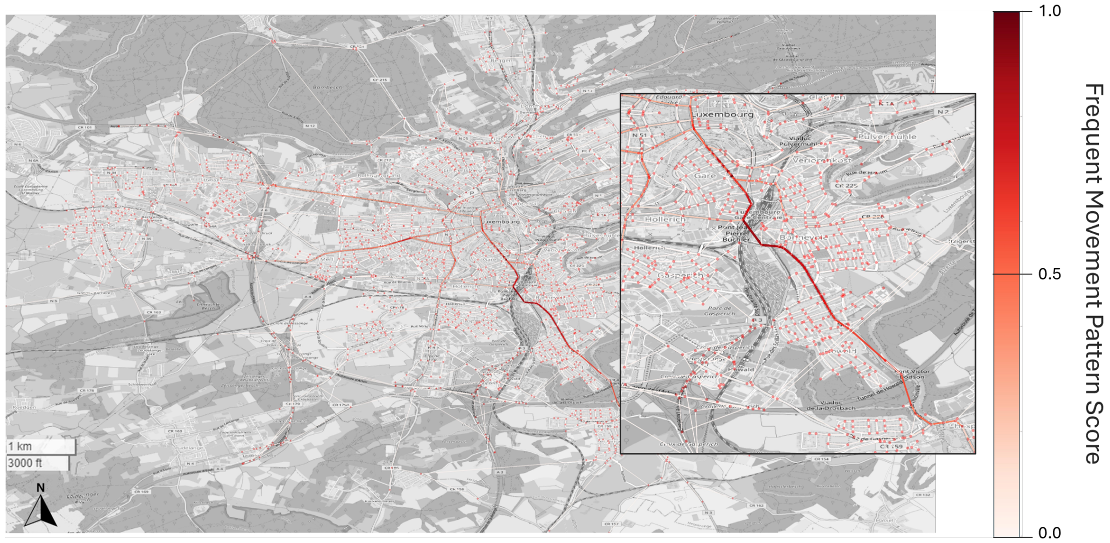

Figure 9: FqM Score

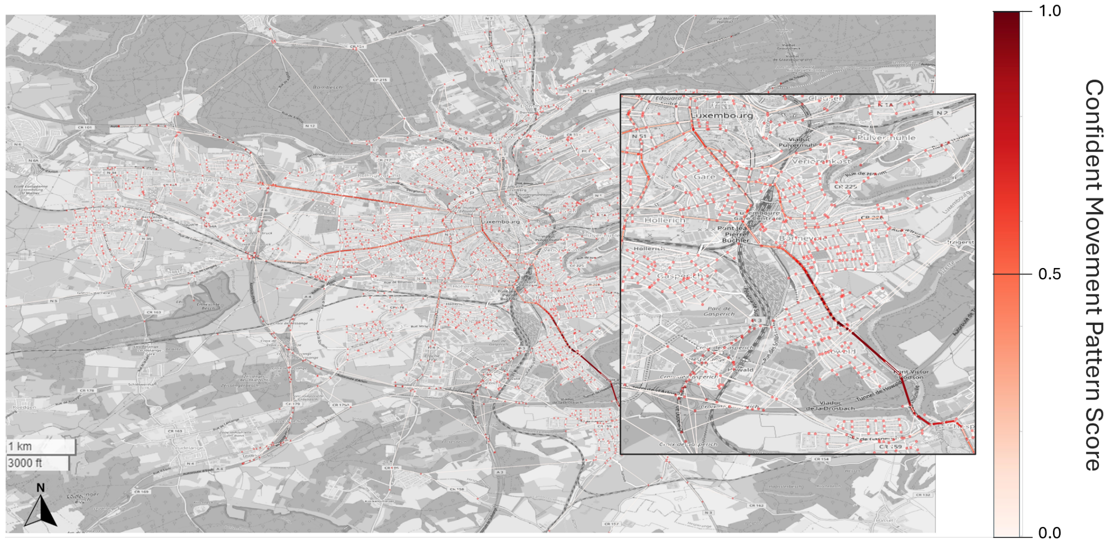

Figure 10: CM Score

frequent movement sequences. Consequently, the visualization effectively highlights prominent roads within the city center, while also capturing the popularity of highways.

Conversely, the visualization of the impact score calculated through CM, as depicted in Figure 10, sheds light on frequently traversed edges within the most confident patterns. Here, a minimum confidence threshold of 0.8 ensures that there is an 80% probability of accessing these edges within a frequent pattern. Consequently, this visualization prioritizes highways and connecting edges, showcasing its preference for routes with higher confidence levels. It is noteworthy that links in the city center, which are prominently scored in FqM, receive a comparatively lower score in CM. This disparity stems from the city center environment, where the presence of alternative routes is more likely.

The introduced innovative metric, the Sequential Impact Score, integrates the insights from both FqM and CM, thereby encapsulating distinct aspects covered by each set. The goal is to highlight critical links—those that are frequently traveled within frequent movement patterns, as well as those with a higher confidence level of being traveled.

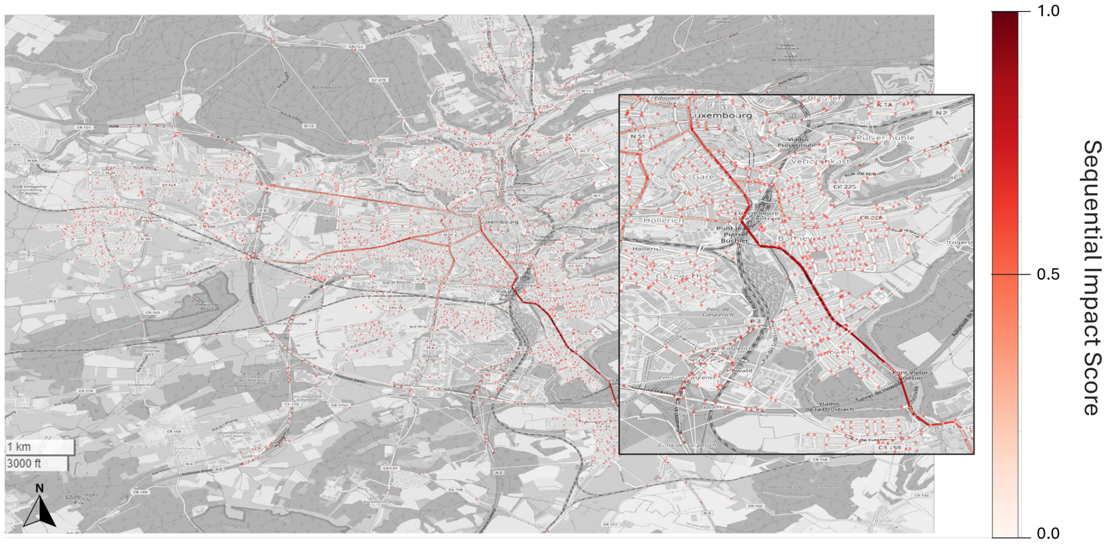

Figure 11: Sequential Impact Score

In the event of disruptions, links identified by the proposed metric are crucial, as they represent areas where alternative paths are either absent or have a low probability (20%) of existence. This comprehensive approach ensures a nuanced understanding of the significance of different links in the context of movement patterns and their potential impact on the overall network.

6. Conclusion and Future Work

In this paper, VeTraSPM is introduced, a novel algorithm for identifying critical road links in urban traffic networks through sequential pattern mining on vehicle trajectory data. The experimental results demonstrate that VeTraSPM provides significant improvements in critical link analysis compared to existing algorithms. By leveraging efficient data structures, parallel processing, and other optimization techniques, the algorithm achieves both high scalability and computational efficiency, making it suitable for large-scale urban traffic networks. The use of the Sequential Impact Score (SIS) provides a novel approach to assessing road criticality. However, future work is required to validate SIS in different traffic conditions and geographical locations.

VeTraSPM is designed to generalize across different urban contexts due to its flexibility in handling diverse trajectory data, making it adaptable to different geographical locations and time spans. However, the algorithm relies on certain assumptions about traffic patterns, which may not apply in areas with unpredictable traffic flows or highly erratic behavior. Additionally, VeTraSPM's current design is limited in its adaptability to real-time disruptions, such as accidents, events, or construction. Incorporating dynamic data streams from real-time traffic sensors would enhance its ability to respond to such changes.

Future work should explore integrating real-time traffic data feeds to enhance the algorithm's responsiveness to sudden traffic disruptions. Additional factors like weather conditions and driver behavior should be incorporated to provide a more comprehensive understanding of traffic patterns. Moreover, the algorithm could be extended to other contexts, such as pedestrian traffic or logistics networks, to broaden its applicability.

In practical terms, VeTraSPM has the potential to assist urban planners and traffic authorities in optimizing traffic management by identifying bottlenecks and critical road segments. The results generated by VeTraSPM can be visualized through tools such as heat maps and interactive dashboards, providing actionable insights for traffic managers and urban planners. VeTraSPM can be integrated into existing traffic management systems for continuous monitoring and planning of infrastructure improvements. With future modifications, it can serve as a valuable tool for real-time traffic monitoring and infrastructure planning, ultimately contributing to more efficient and resilient urban transportation systems.

References

[1] L. Codecá, R. Frank, S. Faye, T. Engel, Luxembourg SUMO Traffic (LuST) Scenario: Traffic Demand Evaluation, IEEE Intelligent Transportation Systems Magazine 9 (2017) 52–63.

[2] R. Agrawal, R. Srikant, Fast algorithms for mining association rules, Proc. 20th Int. Conf. Very Large Data Bases VLDB 1215 (2000).

[3] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, M. Hsu, Freespan: Frequent pattern-projected sequential pattern mining, volume 6, 2000, pp. 355–359. doi:10.1145/347090.347167.

[4] J. Han, J. Pei, X.-F. Yan, From sequential pattern mining to structured pattern mining: A pattern-growth approach, Journal of Computer Science and Technology 19 (2004) 257–279.

[5] F. Thabtah, A review of associative classification mining, Knowledge Eng. Review 22 (2007) 37–65.

[6] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, M.-C. Hsu, Mining sequential patterns by pattern-growth: The prefixspan approach, Knowledge and Data Engineering, IEEE Transactions on 16 (2004) 1424– 1440.

[7] F. Xue, Z. Shan, L.-j. Yan, C. Fan, A improved sequential pattern mining algorithm based on prefixspan, 2016, pp. 1–4. doi:10.1109/WAC.2016.7583059.

[8] P. Fournier Viger, A. Gomariz, T. Gueniche, E. Mwamikazi, R. Thomas, Tks: Efficient mining of top-k sequential patterns, 2013, pp. 109–120. doi:10.1007/978-3-642-53914-5_10.

[9] A. Kemmar, Y. Lebbah, S. Loudni, P. Boizumault, T. Charnois, Prefix-projection global constraint and top-k approach for sequential pattern mining, Constraints 22 (2017).

[10] M. Garofalakis, R. Rastogi, K. Shim, Mining sequential patterns with regular expression constraints, Knowledge and Data Engineering, IEEE Transactions on 14 (2002) 530–552.

[11] K. Oza, D. Kawade, Frequent sequential pattern mining with weighted regular expression and length constraint, International Journal of Scientific Research 4 (2015) 3–7.

[12] J. Wang, J. Han, Bide: Efficient mining of frequent closed sequences, 2004, pp. 79– 90. doi:10.1109/ICDE.2004.1319986.

[13] P. Tzvetkov, X. Yan, J. Han, Tsp: Mining top-k closed sequential patterns, Knowledge and Information Systems 7 (2003).

[14] F. Masseglia, P. Poncelet, M. Teisseire, Incremental mining of sequential patterns in large databases., Data Knowl. Eng. 46 (2003) 97–121.

[15] J. Ayres, J. Flannick, J. Gehrke, T. Yiu, Sequential pattern mining using a bitmap representation, 2002, pp. 429–435. doi:10.1145/775107.775109.

[16] M. Zaki, Zaki, m.j.: Spade: An efficient algorithm for mining frequent sequences. machine learning 42(1), 31-60, Machine Learning 42 (2001) 31–60.

[17] P. Fournier Viger, Fast vertical sequential pattern mining using co-occurrence information., 2014.

[18] A. Gomariz, M. Campos, R. Marín, B. Goethals, Clasp: An efficient algorithm for mining frequent closed sequences, 2013, pp. 50–61. doi:10.1007/978-3-642-37453-1_5.

[19] P. Fournier Viger, C.-W. Wu, A. Gomariz, V. Tseng, Vmsp: Efficient vertical mining of maximal sequential patterns, 2014. doi:10.1007/978-3-319-06483-3_8.

[20] J. Chang, Mining weighted sequential patterns in a sequence database with a time-interval weight, Knowledge-Based Systems 24 (2011) 1–9.

[21] W. Yu, Discovering frequent movement paths from taxi trajectory data using spatially embedded networks and association rules, IEEE Transactions on Intelligent Transportation Systems 20 (2019) 855–866.

[22] R. Ibrahim, M. O. Shafiq, Detecting taxi movements using random swap clustering and sequential pattern mining, Journal of Big Data 6 (2019) 39.

[23] Y. Wang, Y. Tian, B. Yang, J. Wang, X. Hu, S. An, Planning flexible bus service as an alternative to suspended bicycle-sharing service: A data-driven approach, Journal of Advanced Transportation 2023 (2023) 1–15.

[24] S. Hu, Q. Liang, H. Qian, J. Weng, W. Zhou, P. Lin, Frequent-pattern growth algorithm based association rule mining method of public transport travel stability, International Journal of Sustainable Transportation 15 (2020).

[25] L. Moreira-Matias, C. Ferreira, J. Gama, J. Moreira, J. Sousa, Bus bunching detection: A sequence mining approach, volume 960, 2012.

[26] H. Zhang, L. He, Data mining method of sequential patterns for vehicle trajectory prediction in vanet, Wireless Personal Communications 117 (2021) 1–13.

[27] W. Qi, Q. Song, X. Wang, L. Guo, Trajectory data mining-based routing in dtn-enabled vehicular ad hoc networks, IEEE Access PP (2017) 1–1.

[28] A. F. Merah, S. Samarah, A. Boukerche, A. Mammeri, A sequential patterns data mining approach towards vehicular route prediction in vanets, Mobile Networks and Applications 18 (2013) 788–802.

[29] A. F. Merah, S. Samarah, A. Boukerche, Vehicular movement patterns: A prediction-based route discovery technique for vanets, 2012, pp. 5291–5295. doi:10.1109/ICC.2012.6364141.

[30] L. Codeca, J. Härri, Monaco SUMO Traffic (MoST) Scenario: A 3D Mobility Scenario for Cooperative ITS, in: SUMO 2018, SUMO User Conference, Simulating Autonomous and Intermodal Transport Systems, May 14-16, 2018, Berlin, Germany, Berlin, GERMANY, 2018.