Geoinformatica (2017) 21:829–866
DOI 10.1007/s10707-016-0289-3

Discovering non-compliant window co-occurrence patterns

Reem Y. Ali[1] · **Venkata M. V. Gunturi**[2] · **Andrew J. Kotz**[3] · **Emre Eftelioglu**[1] · **Shashi Shekhar**[1] · **William F. Northrop**[3]

Received: 29 April 2016 / Revised: 30 October 2016 / Accepted: 20 December 2016 /
Published online: 30 January 2017

Abstract Given a set of trajectories annotated with measurements of physical variables, the problem of Non-compliant Window Co-occurrence (NWC) pattern discovery aims to determine temporal signatures in the explanatory variables which are highly associated with windows of undesirable behavior in a target variable. NWC discovery is important for societal applications such as eco-friendly transportation (e.g. identifying engine signatures leading to high greenhouse gas emissions). Challenges of designing a scalable algorithm for NWC discovery include the non-monotonicity of popular spatio-temporal statistical interest measures of association such as the cross-K function which renders the anti-monotone pruning based algorithms (e.g. Apriori) inapplicable for such interest measures. In our preliminary work, we proposed two upper bounds for the cross-K function and a top-down multi-parent tracking approach that uses these bounds for filtering out uninteresting candidate patterns and then applies a minimum support (i.e. frequency) threshold as a post-processing step to filter out chance patterns. In this paper, we propose a novel bi-directional pruning approach (BDNMiner) that combines top-down pruning based on the cross-K function threshold with bottom-up pruning based on the minimum support threshold to efficiently mine NWC patterns. Case studies with real world engine data demonstrates the ability of the proposed approach to discover patterns which are interesting to engine scientists. Experimental evaluation on real-world data show that the proposed approach yields substantial computational savings compared to prior work.

Keywords Co-occurrence patterns · Temporal data mining · Transportation

✉ Reem Y. Ali
alireem@cs.umn.edu

[1] Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN, 55455, USA

[2] IIIT-Delhi, New Delhi, India

[3] Department of Mechanical Engineering, University of Minnesota, Minneapolis, MN, 55455, USA

1 Introduction

Given a set of trajectories annotated with measurements of physical variables, the Non-compliant Window Co-occurrence (NWC) pattern discovery problem aims to determine temporal signatures in the explanatory variables which are highly associated with windows of undesirable behavior in a target variable (e.g. non-compliance with some standard). For instance, consider Fig. 1, which shows portions of trajectories of a metro transit bus in Minneapolis-St. Paul, MN, USA. In these trajectories, each point is annotated with physical measurements such as engine power, engine revolutions per minute (RPM), wheel speed, elevation and engine emissions. The red color marks temporal windows within the trajectories where a target variable (emissions of oxides of nitrogen (NO_x) in this example) shows a non-compliant behavior (i.e., the average emissions within the windows exceed US EPA regulations [8]). As shown in the figure, some journeys show this non-compliant behavior, while others do not. NWC discovery aims to determine the underlying temporal signatures of the measured physical variables which are highly associated with those windows of elevated NO_x emissions. These signatures (aka "patterns") represent sequences defined on one or more physical variables that either coincide with or occur within a prespecified time lag from a non-compliant window. Figure 2 shows a non-compliant window <2,6>, of length 5 sec, in which the NO_x emissions exceed the US EPA standard of 0.267 gm/kW-h. As shown in the figure, patterns of high acceleration and increase in elevation co-occur with this non-compliance behavior and thus would be considered as candidate patterns by the NWC discovery problem.

Illustrative application domain Discovering NWC patterns is important to several scientific and societal applications such as eco-friendly transportation (e.g. discovering engine behaviors leading to high greenhouse gas emissions), detection of engine signatures associated with engine malfunctions (e.g. patterns of sudden unintended acceleration [32]) which can help save people's lives, and industrial process control (e.g. understanding patterns of failure in an industrial process [31]). In this paper we use eco-friendly transportation as our illustrative application domain. Current efforts in the field of engine research are aimed at reducing harmful vehicle emissions such as NO_x and carbon dioxide (CO_2) due to their adverse effects on human health and the environment [27, 30]. Despite recent advances

Fig. 1 Non-compliant emissions of oxides of nitrogen along a bus route in Minneapolis, MN (best viewed in color)

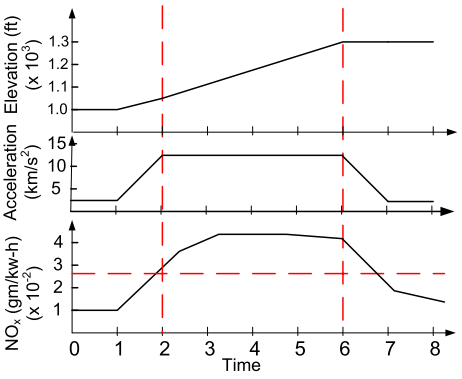


Fig. 2 Example candidate Non-compliant Window Co-occurrence patterns (best viewed in color)

in emissions reduction technologies and stricter standards imposed by regulatory agencies, vehicles are emitting at rates higher than their certified limit under real world driving conditions [20, 21]. However, these discrepancies are not a result of vehicles failing certification. In some instances vehicles have been found using emissions cheating devices [23], but the majority of the discrepancies are a result of a certification test not accurately reflecting real-world vehicle use. Therefore, identifying engine variable signatures co-occurring with elevated NO_x and CO_2 emissions in the real-world is key to understanding the cause of the excess emissions. Furthermore, the ability of the NWC discovery method to discover less frequent patterns from large datasets would be well suited for identifying vehicles which contain emissions defeat devices by highlighting instances where the cheating emissions reduction systems were turned on. Additionally, the time lag in co-occurrence patterns must be considered when analyzing vehicle systems due to the large timescale of some vehicle interactions. For instance, as engine load increases, engine temperature (a key factor in NO_x production) may rise at a slower rate due to the heat capacity and inertia of the engine and coolant and hence the non-compliant NO_x emissions might not occur until a few seconds after the engine load started to increase. Due to the extremely short timescale of combustion and the high frequency of its occurrence, most engine variables vary on a very small timescale except for temperature variables which could take a few seconds to a couple minutes depending on the temperature gradient. Therefore, when studying small fixed non-compliant windows (i.e. a 5 sec window length), it is safe to assume that larger scale temperature changes had already occurred, while other explanatory variables and smaller scale temperature changes exhibit change on a small time scale with no lag between them. In addition, engine measurement datasets are usually collected at an aggregate of a higher frequency (i.e. the actual observations usually happen on the 10-50 Hz timescale while the data is being presented on the 1 Hz timescale). Thus, changes in engine variables will usually happen simultaneously (except for bulk temperature changes which may have a longer time scale) which means that interesting changes in explanatory variables usually occur within and along the non-compliant window and the specified lag value.

To measure the strength of an association between a pattern and non-compliant windows, a spatio-temporal statistical measure is preferred in order to provide a statistical interpretation of the output patterns. The cross-K function [9, 13] is a popular spatio-temporal statistical measure which is often used to measure the interaction between pairs of events in space and time. The cross-K function can express how much the association between a given pattern and non-compliant windows deviates from the assumption of their independence. Additionally, the cross-K function is not sensitive to the prevalence of the output pattern

and hence can capture signatures that are less frequent but are highly associated with non-compliant windows. In addition, a low minimum support (i.e. frequency) threshold can still be used to filter out chance or spurious patterns (e.g., patterns which might have occurred only once in the input data and that occurrence happened to be near a non-compliant window, leading to a misleading high value for its cross-K function). Therefore, using both the cross-K function threshold and a low minimum support threshold allows capturing non-spurious engine signatures that are highly associated with non-compliant windows even when they are not prevalent in the input dataset.

Challenges Designing an algorithm for NWC discovery that captures statistically meaningful patterns while maintaining computational scalability is challenging for the following reasons: First, domain-preferred spatio-temporal statistical association measures (e.g., cross-K function) lack monotonicity: a pattern representing an engine signature over multiple variables may be interesting even though its component single-variable signatures are not. For instance, the increase of both engine RPM and brake torque might be more strongly associated with increased NO_x than the increase of engine RPM alone. This property renders Apriori-based pruning inapplicable for such interest measure. Second, there are a huge number of candidate patterns to consider. For each non-compliant window, the number of associated candidate patterns is exponential in the number of variables. This includes all combinations of one, two, three, etc., variables. Third, the data volume is potentially huge due to the large number of variables over a long time series.

Limitations of related work Related work for the NWC discovery problem mainly consists of literature on mining multi-dimensional temporal association rules [6, 14, 22, 24]. In these rules, a consequent occurs within T time points of an antecedent (a single or multi-dimensional sequence). However, these works, similar to frequent pattern mining [1, 25] have mainly focused on finding the most frequent patterns using a minimum support threshold and Apriori-like pruning methods that rely on the anti-monotone property of the support measure. By contrast, in the NWC discovery problem, rare associations can still be interesting since they can reveal patterns that are highly associated with non-compliant windows but have low support. The statistical interest measure used to capture these patterns does not have this anti-monotone property and hence Apriori-based pruning cannot be applied for this interest measure.

Some other studies in the literature have also addressed the problem of mining rare (i.e. low support) co-occurrence/association patterns with a high confidence threshold [5, 15, 18, 33]. However, these methods only focus on associations between single events and do not model associations between contiguous sequences (e.g., a temporal signature of engine variables co-occurring with non-compliant windows). For instance, they would not be able to capture the association of a continuous acceleration or braking pattern with a window of elevated emissions as shown in our case study. In addition, none of these methods except [15] guarantees completeness. Table 1 shows a classification of the related work.

In our preliminary work [3], we proposed two upper bounds for the cross-K function which are cheaper to compute than the computation of the exact cross-K function. We also proposed a top-down Multi-parent Tracking approach for mining NWC patterns (MTNMiner) that uses the proposed upper bounds for pruning uninteresting NWC patterns. MTNMiner was experimentally validated and a case study was provided for showing its ability to find meaningful patterns. This paper extends our previous work by proposing a

Table 1 Related work of the NWC pattern discovery problem

Temporal Co-occurrence/Association Patterns		Interest Measure	
		Frequent (i.e. High Support) Patterns	Low Support High Confidence /Correlation Patterns
Rule Antecedent	A Set or Sub-sequence of Event Types	[1, 25]	[5, 15, 18, 33]
	(Multivariate) Contiguous Sequence	[6, 14, 22, 24]	Our proposed work

novel bi-directional algorithm (BDNMiner) that improves the scalability of our preliminary MTNMiner algorithm by combining cross-K function pruning with minimum support pruning simultaneously. Figure 3 summarizes the differences between the two algorithms.

Contributions This paper makes the following new contributions: (1) We propose a bi-directional approach for mining NWC patterns (BDNMiner) using both cross-K function and minimum support pruning simultaneously. (Section 4.1) (2) We also propose a method for tightening the cross-K function upper bounds in our BDNMiner for efficiently pruning uninteresting patterns. (Section 4.2) (3) We analytically prove the correctness and completeness of BDNMiner. (4) We present a new case study to evaluate the effectiveness of BDNMiner in finding statistically meaningful engine patterns that are associated with non-compliant CO_2 emissions in transit buses. (5) We provide an experimental evaluation using real-world data and show that BDNMiner yields substantial computational savings compared to our preliminary work [3].

Scope and Outline The process of mining association rules from time series data in continuous domains typically consists of first discretizing the time series values and then discovering the interesting associations [14, 22, 24]. In this paper, we focus on the problem of discovering interesting co-occurrence patterns. We do not address the problem of choosing the most suitable discretization technique. Instead, we assume that the discretization intervals for each variable are given as an input to this problem, possibly using representations suggested in [6, 7, 16, 17]. Additionally, in this work we only focus on temporal co-occurrence patterns. Spatial aspects of non-compliant window co-occurrence patterns will be explored more thoroughly in future work. We also only considered using a fixed lag value for all explanatory variables and assumed that all patterns have the same length as the non-compliant windows.

The rest of the paper is organized as follows: Section 2 presents the basic concepts followed by a formal problem definition for the NWC pattern discovery problem. Section 3

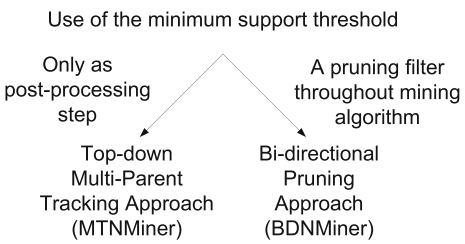


Fig. 3 Our new BDNMiner approach versus our preliminary MTNMiner approach

reviews our preliminary approach towards addressing this problem [3]. The proposed bi-directional BDNMiner algorithm is presented in Section 4. Section 5 presents a theoretical evaluation of the correctness and completeness of BDNMiner. Section 6 presents case studies for using the BDNMiner algorithm on real-world engine datasets collected from transit buses. The experimental evaluation is covered in Section 7. Finally, Section 8 concludes the paper and discusses future work.

2 Basic concepts and problem statement

2.1 Basic concepts

Definition 1 An event: Given a variable v, an event $e_i(v)$ is a reading where v falls within a predefined range $[v_i, v_{i+1})$.

For example, a set of events $E(v) = \{e_1(v), e_2(v), .., e_m(v)\}$ can be defined for the wheel speed variable v where $e_1(v)$ indicates that wheel speed $\in$ [0, 5) km/h, $e_2(v)$ indicates that wheel speed $\in$ [5, 10) km/h, and so on.

Definition 2 A multi-variate event trajectory (MET): Given a set of explanatory variables V and a target variable y, a MET is a sequence of multivariate points $p_t = (p_t^1, p_t^2, ..., p_t^{|V|}, y_t)$, $1 \leq t \leq \tau$, where t is a timestamp of p_t, τ is the trajectory length, p_t^k is an event defined for variable $v_k \in V$, $1 \leq k \leq |V|$, and $y_t \in \mathbb{R}$.

Figure 4 shows an example of a MET of length τ=8, defined over two explanatory variables V={v_1:engine power, v_2:engine RPM}, where $E(v_1)$={a_1, a_2, a_3} and $E(v_2)$={b_1, b_2, b_3} are their corresponding sets of events, and a target variable y of NO_x emissions.

Definition 3 An event-sequence S(v): Given a variable v, an event-sequence $S(v)$ is a sequence of events $e_i(v)$ that are temporally contiguous in a MET.

For example, in Fig. 4, $a_2a_3a_2$ is an event sequence for engine power.

Next, before we present the definition of a non-compliant window, we first need to distinguish between two types of functions, namely, local functions and zonal functions. Local functions are functions that determine the output at each time instant based on the attribute value at this time instant. For example, computing the engine load at a given time instant as a function of the engine power at that time instant is a local function. Zonal functions employ aggregate operators over time instants in a longer interval. For example, determining the average NO_x emissions within a 10 sec window is a zonal function.

Time	0	1	2	3	4	5	6	7
v_1: Engine Power	a_1	a_2	a_3	a_2	a_1	a_2	a_3	a_2
v_2: Engine RPM	b_1	b_1	b_2	b_3	b_1	b_1	b_2	b_2
NO_x (gm/sec)	0.011	0.011	0.015	0.023	0.023	0.021	0.019	0.019

Fig. 4 Input data with one non-compliant window (best in color)

Definition 4 **A non-compliant window (W_N)**: Given a MET m, a zonal function F defined over the target variable y of m and a window length L, and a threshold h, a non-compliant window $W_N =< t_i,t_j >$ is a time interval in m, of length L, where $F(y) > h$. The length L is defined as the number of time instants within the window, i.e. $L = t_j - t_i + 1$.

For example, given the MET in Fig. 4, and a function F defined as the percentage of increase in NO_x between the start and end of a 3 sec window, and a threshold of 100 %, the window <1,3> is a non-compliant window since $\frac{0.023-0.011}{0.011}$ = 109 %> 100 %.

Definition 5 **A Non-compliant Window Co-occurrence (NWC) pattern**: Given a MET m defined over a set of explanatory variables V and a target variable y, and a time lag δ, an NWC pattern C is a set of equal-length event-sequences $\{S_i(u_i) \mid u_i \in U, U \subseteq V$ and $1 \leq i \leq |U|\}$, that started at the same time point, and within a time lag δ preceding the start of a non-compliant window in m. Length(C) denotes the length of the event-sequences in C and is equal to the non-compliant window length L. $Dim(C)$ denotes the dimensionality of pattern C (i.e. number of variables in C), where $Dim(C)= |U|$.

For example, in Fig. 4, given a time lag δ=1 sec, we can identify 6 NWC patterns as listed in Table 3. The first three patterns coincide with the non-compliant window, while the last three patterns precede the window by a lag of 1 sec. Patterns with IDs =1, 2, 4 and 5 have a dimensionality of 1 since they are defined on only one variable, while patterns 3 and 6 have a dimensionality of 2.

2.2 Interest measure: temporal cross-K function

In this work, the temporal cross-K function, a purely temporal form of the space-time cross-K function [9, 13], is used as a statistical measure to express how much the association between a given pattern and non-compliant windows deviates from independence. A temporal cross-K function measuring the association between an NWC pattern, C, and the occurrence of non-compliant windows, W_N, at a time lag δ is calculated as follows: $K_{C,W_N}(\delta) = \lambda_{W_N}^{-1}$ E[number of non-compliant windows starting within time δ from the start of an instance of C], where λ_{W_N} is the expected number of non-compliant window events per unit time. Under the assumption of independence between the occurrences of pattern C and the non-compliant windows, $K_{C,W_N}(\delta)$ is equal to $(\delta + 1)$. Whenever $K_{C,W_N}(\delta)$ is greater than $\delta + 1$, this indicates an association between the pattern and the non-compliant behavior, with higher values indicating a stronger association. According to [10], $K_{C,W_N}(\delta)$ can be estimated by:

$$\begin{aligned}\hat{K}_{C,W_N}(\delta) &= \lambda_{W_N}^{-1} \sum_i \sum_j \frac{I(0 \leq d(C_i, W_{N\,j}) \leq \delta)}{|C|} \\ &= \frac{T}{|W_N||C|} \sum_i \sum_j I(0 \leq d(C_i, W_{N\,j}) \leq \delta) \end{aligned} \quad (1)$$

where $d(C_i, W_{N\,j})$ is the distance between the start of instance C_i of pattern C and the start of the non-compliant window $W_{N\,j}$; I(.) is an indicator function that assumes a value of 1 if $0 \leq d(C_i, W_{N\,j}) \leq \delta$, and a value of 0 otherwise; T= $\sum_{allMETs} \tau$ (and is referred to as

the time series length in this paper), and $|W_N|$ and $|C|$ are, respectively, the number of non-compliant windows and the number of instances (i.e., cardinality) of pattern C across all METs (i.e., the time series). Hence, $\hat{K}_{C,W_N}(\delta)$ can be written as:

$$\hat{K}_{C,W_N}(\delta) = \frac{T \times |C \overset{\delta}{\bowtie} W_N|}{|W_N||C|} \tag{2}$$

where $|C \overset{\delta}{\bowtie} W_N|$ denotes the cardinality of the temporal join set between the instances of pattern C and non-compliant windows, W_N, such that an instance C_i and $W_{N\,j}$ are only joined if C_i preceded $W_{N\,j}$ by time t where $0 \leq t \leq \delta$. For simplicity, in the rest of this paper, we will refer to $|C \overset{\delta}{\bowtie} W_N|$ as the cardinality of the join set of C.

2.3 Problem statement

The problem of discovering Non-compliant Window Co-occurrence (NWC) patterns can be formally expressed as follows:
Given:

1. A set M of multivariate event trajectories (METs)
2. A set of non-compliant windows W_N (i.e. time intervals on M), where each window is of length L
3. A time lag δ,
4. A temporal cross-K function threshold ϵ, and
5. A minimum support threshold $minsupp$

Find: All NWC patterns C where $\hat{K}_{C,W_N}(\delta)$ ¿ ϵ.
Objective: Reduce computational cost.
Constraints:

1. All output patterns should have a support at least equal to $minsupp$.
2. All METs in M are sampled at equal intervals with the same sampling rate.
3. Correctness and completeness are guaranteed.

The $minsupp$ threshold is mainly used to discard chance/spurious patterns and reduce the number of patterns output to the user, however, it should generally be set to a very low value to allow the discovery of both rare as well as frequent patterns. Table 2 summarizes the notations used in the above problem statement.

Table 2 Table of notations

Symbol	Description
M	set of multivariate event trajectories (METs)
W_N	set of non-compliant windows
L	non-compliant window length
δ	time lag
ϵ	temporal cross-K function threshold
$minsupp$	minimum support threshold
C	an NWC pattern
$\hat{K}_{C,W_N}(\delta)$	cross-K function of pattern C at time lag δ

Assumptions: In our above problem formulation, we have made the following simplifying assumptions:
1. All explanatory variables are assumed to have the same lag value δ.
2. The length of a non-compliant window is exactly the same as the candidate patterns length.

Example: Figure 4 shows an example for the input to the NWC discovery problem. The input consists of a MET defined on two explanatory variables: v_1:engine power and v_2:engine RPM, and a target variable: NO_x emissions in gm/sec. $E(v_1) = \{a_1, a_2, a_3\}$ and $E(v_2) = \{b_1, b_2, b_3\}$ are the set of events defined for the engine power and engine RPM variables, respectively. One non-compliant window <1,3> of length $L = 3$ sec is input and is marked by a red rectangle as shown in the figure. Hence, $|W_N| = 1$. The $\hat{K}_{C,W_N}(\delta)$ threshold ϵ is set to 5, δ is set to 1 sec, and $minsupp$ is set to 1/8 (for illustration purposes only). The aim is to find all NWC patterns meeting the $\hat{K}_{C,W_N}(\delta)$ and $minsupp$ thresholds. Table 3 shows all the candidate NWC patterns. The first 3 patterns are those coinciding with the non-compliant window <1,3>, while the next 3 patterns are the patterns preceding the window by a lag of 1 sec. Columns 3, 4, 5 and 6 show the number of occurrences of each pattern in the time series (i.e. pattern cardinality), the cardinality of the join set between instances of this pattern and non-compliant windows, the pattern support, and the value of the interest measure, respectively.

For example, the first pattern in the table occurred twice at time instants 1 and 5 (i.e. $|C|$=2). That is, it has a support = $\frac{|C|}{T}$ = 2/8 = 1/4. However, only one of those occurrences was associated with a non-compliant window: the pattern instance at t=1 coincided with the non-compliant window <1,3> (i.e. $|C \overset{1}{\bowtie} W_N|$=1). Hence, for this pattern, $\hat{K}_{C,W_N}(\delta) = \hat{K}_{C,W_N}(1) = \frac{T \times |C \overset{1}{\bowtie} W_N|}{|W_N||C|} = \frac{8 \times 1}{1 \times 2} = 4 < 5$. However, the second pattern occurred only once at t=1, where it coincided with a non-compliant window. Hence for this pattern, $\hat{K}_{C,W_N}(1) = \frac{T \times |C \overset{1}{\bowtie} W_N|}{|W_N||C|} = \frac{8 \times 1}{1 \times 1} = 8 > 5$ and support(C) = 1/8 $\geq minsupp$. As shown in Table 3, only patterns 2 and 3 have an interest measure (cross-K function) exceeding ϵ and support at least equal to $minsupp$, and thus these are the final output patterns as indicated in column 7.

3 Preliminary results

In this section, we first present the naive approach for solving the NWC discovery problem [3]. Then, we review our preliminary **M**ulti-Parent **T**racking approach for mining **N**WC patterns (MTNMiner) with its cross-K function upper bound pruning filters.

Table 3 Candidate NWC patterns for the input in Fig. 4

ID	Candidate Pattern C	| C |	$\lvert C \overset{\delta}{\bowtie} W_N \rvert$	support(C)	$\hat{K}_{C,W_N}(1)$	Is output?
1	$\{a_2a_3a_2\}$	2	1	1/4	4	NO (4 <5)
2	$\{b_1b_2b_3\}$	1	1	1/8	8	YES (8 > 5)
3	$\{a_2a_3a_2, b_1b_2b_3\}$	1	1	1/8	8	YES (8 > 5)
4	$\{a_1a_2a_3\}$	2	1	1/4	4	NO (4 <5)
5	$\{b_1b_1b_2\}$	2	1	1/4	4	NO (4 <5)
6	$\{a_1a_2a_3, b_1b_1b_2\}$	2	1	1/4	4	NO (4 <5)

3.1 Naive approach

The naive approach starts by finding all non-compliant windows in the given time series (i.e. the collection of input METs) using a sliding window of the same length as the given non-compliant window length. Then, for each non-compliant window in a MET m, we enumerate all temporal windows in the MET that started within time t preceding this non-compliant window, where $0 \leq t \leq \delta$. Finally, for each of these temporal windows we enumerate all the candidate NWC patterns. Each pattern C is enumerated by calculating its cardinality $|C|$ using a single linear scan of the time series. Whenever an instance of the pattern is found, the algorithm examines the non-compliant windows table to count the number of windows that are within δ sec from this pattern. Hence, for a pattern C, both $|C|$ and $|C \overset{\delta}{\bowtie} W_N|$ are calculated using a single linear scan. Finally, if the pattern satisfies the $minsupp$ threshold, its cross-K function is calculated and the pattern is output if the measure exceeds the user-specified threshold ϵ.

Note that while enumerating the non-compliant windows and their corresponding candidate patterns, no non-compliant window or pattern is allowed to overlap two different METs. In addition, if the input METs belong to different moving objects (e.g. different vehicles), NWC patterns in a MET of one object should not be associated with a non-compliant window in a MET of another object. To achieve this, the non-compliant window table also stores the object ID to differentiate between the non-compliant windows of the different objects.

3.2 Key ideas behind MTNMiner

In this subsection, we review the three key ideas behind the MTNMiner algorithm before presenting the algorithm in the following subsection.

Key Idea 1: Local Upper Bound of $\hat{K}_{C,W_N}(t)$:

First, we define a subset/superset relation between NWC patterns.

Definition 6 **Subset and superset patterns**: Given two NWC patterns, C=$\{S_i(u_i) : u_i \in$ U, $1 \leq i \leq |U|\}$ and C'=$\{S_i(q_i) : q_i \in$ Q, $1 \leq i \leq |Q|\}$, then, C' is said to be a **subset** of C iff: (1)Length(C') = Length(C). (2) Q $\subseteq$ U. (3) For every $S_i(q_i) \in$ C', we have $S_i(q_i) \in$ C. Similarly, C is said to be a **superset** of C' (i.e. superset(C')).

Definition 7 **Local upper bound**: Given an NWC pattern C=$\{S_i(u_i) \mid u_i \in$ U, U $\subseteq$ V and $1 \leq i \leq |U|\}$ and a time lag δ, the **local upper bound** of $\hat{K}_{C,W_N}(\delta)$, denoted as $UB_{local}(\hat{K}_{C,W_N}(\delta))$, can be computed as follows:

$$
\begin{aligned}
UB_{local}(\hat{K}_{C,W_N}(\delta)) &= \frac{T}{|W_N|} \times \frac{Upper_{Loc}(|C \overset{\delta}{\bowtie} W_N|)}{Lower(|C|)} \\
\text{where: } Upper_{Loc}(|C \overset{\delta}{\bowtie} W_N|) &= \min_{\{S_i\} \in C, 1 \leq i \leq Dim(C)} (|\{S_i\} \overset{\delta}{\bowtie} W_N|) \\
\text{and } Lower(|C|) &= |\text{superset(C)}| \qquad (3)
\end{aligned}
$$

Note that $Upper_{Loc}(|C \overset{\delta}{\bowtie} W_N|)$ is an upper bound of $|C \overset{\delta}{\bowtie} W_N|$ which exists in the numerator of $\hat{K}_{C,W_N}(\delta)$. It is computed using the minimum join set cardinality of all subset patterns of C that consist of only one event-sequence (i.e. one-variable subset patterns).

$Lower(|C|)$ is a lower bound of $|C|$, which is in the denominator of $\hat{K}_{C,W_N}(\delta)$, and is equal to the cardinality of any superset pattern of C. Next, we prove that $UB_{local}(\hat{K}_{C,W_N}(\delta))$ is an upper bound of $\hat{K}_{C,W_N}(\delta)$.

Lemma 1 *Given an NWC pattern C and a time lag δ,* $Upper_{Loc}(|C \overset{\delta}{\bowtie} W_N|)$ *is an upper bound of* $|C \overset{\delta}{\bowtie} W_N|$

Lemma 2 *Given an NWC pattern C and a time lag δ,* $Lower(|C|)$ *is a lower bound of* $|C|$.

Theorem 1 *Given an NWC pattern C and a time lag δ,* $UB_{local}(\hat{K}_{C,W_N}(\delta))$ *is an upper bound of* $\hat{K}_{C,W_N}(\delta)$.

Proofs of Lemmas 1 and 2 and Theorem 1 can be found in Appendix A and in [3]. Now, since $UB_{local}(\hat{K}_{C,W_N}(\delta))$ is an upper bound of $\hat{K}_{C,W_N}(\delta)$, then if this upper bound is less than the cross-K function threshold ϵ, pattern C will not be output and hence there is no need to compute the actual cardinality of the pattern or the cardinality of its join set.

Key Idea 2: Lattice Upper Bound of $\hat{K}_{C,W_N}(t)$:

We use a second upper bound for the $\hat{K}_{C,W_N}(\delta)$ of a pattern C, denoted as $UB_{lattice}(\hat{K}_{C,W_N}(\delta))$. Although this bound is less tight than the local upper bound of a pattern, $UB_{lattice}(\hat{K}_{C,W_N}(\delta))$ has a conditional monotone property. Based on that property, if $UB_{lattice}(\hat{K}_{C,W_N}(\delta))$ is less than ϵ, then the lattice upper bound for all subset patterns of C is also less than ϵ and so they can be completely pruned without calculating their upper bounds.

Definition 8 Lattice upper bound: Given an NWC pattern C=$\{S_i(u_i) \mid u_i \in U, U \subseteq V$ and $1 \leq i \leq |U|\}$ and a time lag δ, the **lattice upper bound** of $\hat{K}_{C,W_N}(lag)$, denoted as $UB_{lattice}(\hat{K}_{C,W_N}(\delta))$, can be computed as follows:

$$
\begin{aligned}
UB_{lattice}(\hat{K}_{C,W_N}(\delta)) &= \frac{T}{|W_N|} \times \frac{Upper_{Lat}(|C \overset{\delta}{\bowtie} W_N|)}{Lower(|C|)} \\
\text{where: } Upper_{Lat}(|C \overset{\delta}{\bowtie} W_N|) &= \max_{\{S_i\} \in C, 1 \leq i \leq Dim(C)} (|\{S_i\} \overset{\delta}{\bowtie} W_N|) \\
\text{and } Lower(|C|) &= |\text{superset(C)}|
\end{aligned} \tag{4}
$$

Theorem 2 *Given an NWC pattern C and a time lag δ,* $UB_{lattice}(\hat{K}_{C,W_N}(\delta))$ *is an upper bound of* $\hat{K}_{C,W_N}(\delta)$.

Conditional Monotone Property for the Lattice Upper Bound

Next we present Lemma 3 which describes a conditional monotone property for the lattice upper bound. Proofs of Theorem 2 and Lemma 3 can also be found in Appendix A and in [3].

Lemma 3 *Given an NWC pattern C and a time lag δ,* $UB_{lattice}(\hat{K}_{C,W_N}(\delta))$ *is monotonically decreasing with decreasing Dim(C) if* $Lower(|C|)$ *is kept monotonically increasing. In other words, given two NWC patterns C and C' where C' ⊂ C, then if* $Lower(|C'|) \geq Lower(|C|)$, *then* $UB_{lattice}(\hat{K}_{C',W_N}(\delta)) \leq UB_{lattice}(\hat{K}_{C,W_N}(\delta))$.

Key Idea 3: Efficient Calculation of the Pattern Cardinality:
A more efficient method to calculate the pattern cardinality is to preprocess the time series to create a $startingEdge$ index. This index is a hash table where the key is two events that occurred consecutively in time i.e., $s_1 \rightarrow s_2$. The value is a list of all the time instants where this edge appeared in the input time series. A separate index is kept for each of the input variables. To calculate the cardinality of a pattern, we use the first two consecutive events (i.e., the first edge in the pattern) as the key, and retrieve the corresponding time instants where this edge occurred from the hash table. Then, we only search the time series at these time instants to count the cardinality of the pattern.

3.3 MTNMiner: A Multi-Parent Tracking Approach for Mining NWC patterns

The MTNMiner algorithm starts by finding and then iterating through all non-compliant windows. For each window, it enumerates patterns starting at t time points preceding that window, where $0 \leq t \leq \delta$. In addition, MTNMiner uses the key ideas introduced in the previous subsection to efficiently traverse the candidate patterns enumeration space. For each value of t preceding a non-compliant window $< t_i, t_j >$, a lattice data structure is used to represent all the patterns starting at t, as shown in Fig. 6a. The lattice nodes represent all the possible patterns within the window $< t_i - t, t_j - t >$. Each node is labeled with the list of variables in the pattern it belongs to. For example, consider the input MET shown in Fig. 5. For the window $< 0, 2 >$, the lattice node labelled $\{a, b\}$ represents the pattern defined by the first two variables in that window, namely $\{a_1a_2a_3, b_1b_1b_2\}$.

Within the lattice of each window, MTNMiner starts by enumerating all leaf nodes representing one-variable patterns, and stores the join-set cardinality of these nodes in an array $LeafJoinSetCount$. Then, a top-down breadth first traversal/search (BFS) is performed to enumerate the rest of the lattice nodes while applying the previously defined upper bounds. Since each node has multiple parents, a node can be pruned through the lattice upper bound of any of its parent nodes. Therefore, a node is inserted into the BFS queue for enumeration only if all its parents were already visited and none had a lattice upper bound $> \epsilon$. Hence, each node keeps track of the number of its unvisited parents (i.e. $unVisitedParents$). This also avoids adding duplicate copies of a node to the queue through the node's multiple parents. In addition, each node stores the following information: (1)$supersetCount$: the maximum cardinality found so far of a superset pattern of this node; and (2)$isPruned$: a flag to indicate if the node was already pruned through one of its ancestor nodes. Initially, for each node n, $isPruned$ is set to $False$, $unVisitedParents$ is set to the number of parent nodes of n, and $supersetCount$ is set to 1 since we are sure that there is at least one instance of the root node pattern in the current window, and this root node pattern is a superset of all the patterns in that window. An $enumeratedPatterns$ table is used to store the patterns already enumerated. This table also stores the cardinalities of the leaf nodes' patterns and their join sets. Finally, the algorithm uses a queue to perform a Breadth First Traversal for the lattice nodes.

Time	0	1	2	3	4	5	6	7	8	9	10	11
var_a	a_1	a_2	a_3	a_2	a_1	a_2	a_3	a_2	a_1	a_2	a_3	a_2
var_b	b_1	b_1	b_2	b_3	b_1	b_1	b_2	b_2	b_1	b_2	b_3	b_2
var_c	c_1	c_2	c_1	c_1	c_1	c_2	c_1	c_1	c_2	c_1	c_2	c_2
var_d	d_2	d_2	d_3	d_3	d_2	d_2	d_3	d_2	d_1	d_2	d_2	d_3

Fig. 5 Input with two non-compliant windows

Algorithm 1 MTNMiner

```
1: enumeratedPatterns ← {}
2: Queue queue ← {}
3: startingEdgeIndex ← CreateStartingEdgeIndexFromMETs
4: lattice ← Create and initialize lattice
5: for each window w=<t_i, t_j>in W_N do                    ▷ iterate through all non-compliant windows
6:     for t := δ to 0 do                                    ▷ iterate from 0 to the max lag δ preceding w
7:         latticeCp ← CreateDeepCopy(lattice)
8:         LeafJoinSetCount ← EnumerateOneVariableNodes(latticeCp,enumeratedPatterns)
9:         queue.enqueue(latticeCp.root)
10:        while queue not empty do
11:            Node node ← queue.dequeue()
12:            enumerateWithUpperBoundPruning(latticeCp,node,queue,w,t,δ)

13: function enumerateWithUpperBoundPruning(lattice,n,queue,w,t,δ)
14:    if UB_lattice(K̂_{n.C,W_N}(δ)) ≤ ε then PruneAllNodeSubsets(n,lattice)
15:    else if UB_local(K̂_{n.C,W_N}(δ)) ≤ ε then
16:        for each unpruned non-leaf child node ch of n do
17:            ch.supersetCount ← max(ch.supersetcount,n.supersetcount)
18:            Check if n is last visited parent of ch, then queue.enqueue(ch)
19:    else                                                 ▷ no pruning occurred
20:        C ← expandPattern(n)
21:        if C not in enumeratedPatterns then
22:            [|C|, |C ⋈^δ W_N|] ← Calculate cardinalities using startingEdgeIndex
23:            enumeratedPatterns.put(C)
24:            if |C|/T ≥ minsupp and K̂_{C,W_N}(δ) >ε then Output C.
25:            for each unpruned non-leaf child node ch of n do
26:                ch.supersetCount ← max(ch.supersetcount,|C|)
27:                Check if n is last visited parent of ch, then queue.enqueue(ch)
28:        else                                             ▷ C already enumerated
29:            PruneAllNodeSubsets(n)
```

Algorithm 1 shows the pseudo code of MTNMiner. First, the algorithm starts by initializing the used data structures (lines 1-4). Next, the pattern enumeration step is performed (lines 5-12). The algorithm iterates through all temporal windows starting within a time lag t preceding a non-compliant window, where $0 \leq t \leq \delta$. For each temporal window, MTNMiner starts by creating a copy of the initial lattice to enumerate the patterns in that window (line 7). Pattern enumeration proceeds in two phases: **Phase 1 (line 8)** enumerates the patterns represented by all the leaf nodes. Each pattern is expanded by retrieving it from the input time series. If the pattern was already enumerated, its cardinality and join set cardinality are retrieved from the $enumeratedPatterns$ table and used to calculate its cross-K function. The join set cardinality is also stored in the $LeafJoinSetCount$ array. Otherwise, the pattern cardinality and its join set cardinality are calculated and stored with the pattern in the $enumeratedPatterns$ table. Additionally, the join set cardinality is stored in the $LeafJoinSetCount$ array to be used in calculating the upper bounds for the rest of the lattice nodes. In **phase 2 (lines 9-12):** the algorithm performs a top-down breadth first traversal starting from the root node of the lattice and continuing until the queue is empty. For each node in the queue, the function $EnumerateWithUpperBoundPruning(.)$ (lines 13-29) is called, as follows.

The $EnumerateWithUpperBoundPruning(.)$ function starts by calculating the lattice upper bound of the node using the maximum join set cardinality of all the one-variable subsets of the node (already stored in $LeafJoinSetCount$) and the $supersetCount$ value of the node. If the lattice upper bound is $\leq \epsilon$, all subset patterns of this node (i.e. all descendant nodes) are marked as pruned (line 14). If not, the local upper bound is calculated. If this bound is $\leq \epsilon$ (line 15), then the cost of enumerating the pattern represented by this node

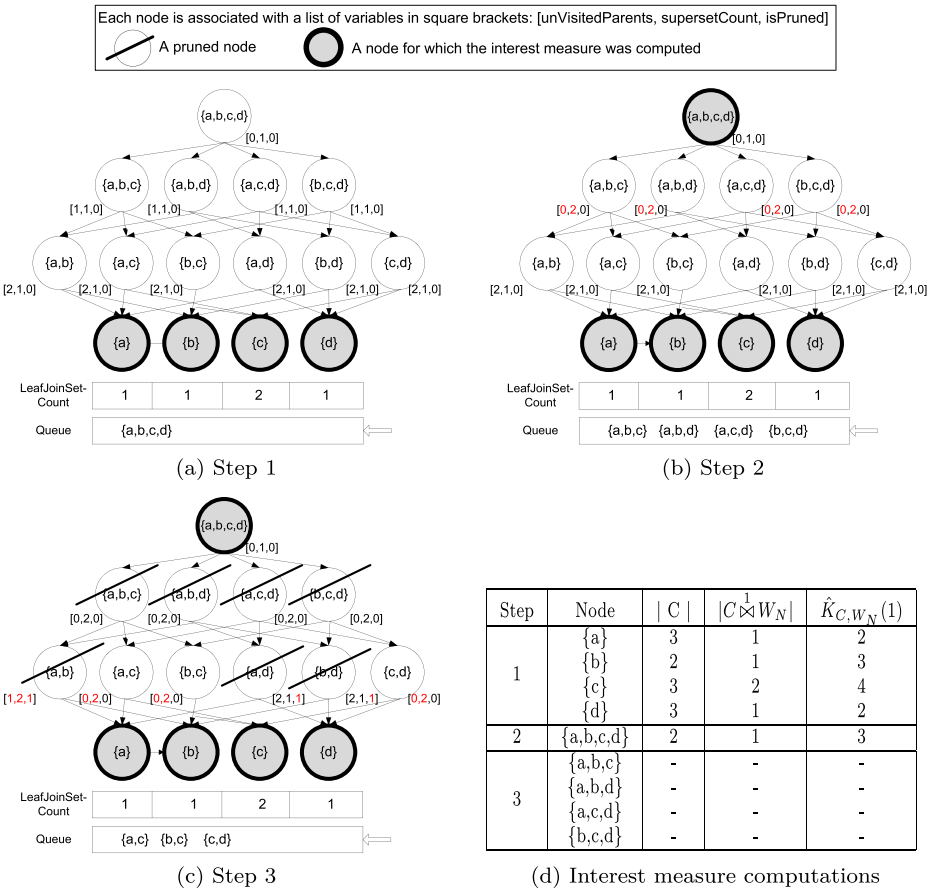
(i.e., expanding it and calculating its cardinality and the cardinality of its join set) is saved. However, we still need to examine the children of this node (lines 16-18). For each child node not marked as pruned, we set its $supersetCount$ variable to the maximum of its current $supersetCount$ value and the $supersetCount$ of its parent node. Then, we decrease the number of $unVisitedParents$ for the child by one and if this was the last visited parent (i.e. unVisistedParents = 0), we insert the child node into the queue. Finally, if the local upper bound of the node was greater than ϵ, then we have to enumerate this node (lines 19-29). First the node is expanded by retrieving the actual pattern from the time series. If the pattern was already enumerated (lines 28-29), all its subset nodes are marked as pruned. Otherwise (lines 21-27), the cardinalities of the pattern and its join set are calculated using the $startingEdge$ index, and the pattern is inserted into the $enumeratedPatterns$ table. If the pattern satisfies the $minsupp$ threshold, its cross-K function is calculated and if this value exceeds ϵ, the pattern is output (line 24). Finally (lines 25-27), the child nodes are treated in the same way as described before; however, in this case, the $supersetCount$ of each child is set to the maximum of its current value and the cardinality computed for the parent node (line 26). As the $supersetCount$ value of each node increases, the lattice upper bound becomes tighter.

Bottleneck analysis Table 4 shows a break-down of the running time of MTNMiner without pruning. A MET with T =50,000 points is used, with L= 5 sec, δ = 1 sec, ϵ = 15, and $minsupp$=0.01 % threshold is specified. As shown in Table 4, the main bottleneck is calculating the cardinality of candidate patterns and their join set. By comparison, the time required for copying and traversing the actual lattice (in addition to all other tasks) is negligible. Hence, our pruning strategies focus on avoiding this cardinality computation cost. It should also be noted that although for each enumerated window, a lattice is created with nodes representing all candidate patterns within that window, only one lattice at a time is kept in memory.

Execution Trace Figure 6 shows an example run of MTNMiner for the input data shown in Fig. 5. The MET is of length T=12 and has two non-compliant windows of length L = 3 sec, namely, <1,3>and <7,9>. The cross-K function threshold is set to ϵ = 3.5, δ = 1 sec and $minsupp$=1/6 (for illustration). For brevity, the execution trace shows only the enumeration of candidates within one window <0,2>, which started 1 sec before the non-compliant window <1,3>. Similar enumerations will be done for the windows <1,3>, <6,8>and <7,9>. Figure 6 shows the lattice for window <0,2>after executing each step of the algorithm, where one whole level is enumerated at every step. Figure 6a shows the lattice created for window <0,2>after enumerating the leaf-nodes and calculating their cross-K function. Their join set cardinalities are shown in the array $LeafJoinSetCount$.

Table 4 MTNMiner bottleneck analysis

No. of variables	Cardinality Counting Time	Other Tasks Time	Total Time
6	267.2 sec	3.1 sec	270.3 sec
8	769.2 sec	9.3 sec	778.5 sec
10	3772 sec	39 sec	3811 sec



Step	Node	| C |	$\lvert C \overset{1}{\bowtie} W_N \rvert$	$\hat{K}_{C,W_N}(1)$
1	{a}	3	1	2
	{b}	2	1	3
	{c}	3	2	4
	{d}	3	1	2
2	{a,b,c,d}	2	1	3
3	{a,b,c}	-	-	-
	{a,b,d}	-	-	-
	{a,c,d}	-	-	-
	{b,c,d}	-	-	-

(d) Interest measure computations

Fig. 6 MTNMiner Execution trace (Best viewed in color)

Figure 6d shows the computed values at step 1, in which only the pattern of node $\{c\}$ is output, namely $\{c_1c_2c_1\}$, since its cross-K function equals $4>\epsilon$ and its support equals $3/12>1/6$. Next, the root node is inserted in the queue. Figure 6b shows the lattice after enumerating the root node by calculating its lattice upper bound ($= \frac{12}{2} \times \frac{\max\{1,1,2,1\}}{1} = 12>\epsilon$) and local upper bound ($= \frac{12}{2} \times \frac{\min\{1,1,2,1\}}{1} = 6>\epsilon$). Since both values exceed ϵ, the actual cardinalities of the root pattern and its join set are calculated. Then for its child nodes, the $unVisitedParents$ variable is decremented, their $supersetCount$ is set to the actual cardinality of the root node pattern just computed (changes are marked in red), and the child nodes are inserted into the queue. In Fig. 6c, the lattice upper bound is calculated for the first node in the queue, $\{a, b, c\}$. Although its value exceeds ϵ, the local upper bound value of 3 is less than ϵ and hence the node is pruned (from Theorem 1). Then, the $supersetCount$ of its child nodes is set to its $supersetCount$ value and their $unVisitedParents$ are decremented. Node $\{a, b, d\}$, which is next in the queue, is enumerated by calculating its lattice upper bound. Since the bound is equal to $3 <\epsilon$, the node is pruned and its three child nodes are also marked as pruned (from Theorem 2 and Lemma 3). The enumeration continues

similarly for all the nodes in this level where nodes $\{a, c, d\}$ and $\{b, c, d\}$ will also be pruned through their local upper bounds (=3<ϵ). Finally, the next level of nodes will be enumerated similarly until the queue is empty.

In the next section, we will present a new proposed approach for NWC pattern mining that enhances the scalability of MTNMiner by combining cross-K function pruning using the lattice and local upper bounds with minimum support pruning.

4 Proposed approach

In this section, we present our proposed **B**i-**D**irectional approach for mining **N**WC patterns (BDNMiner). We start by describing the bi-directional traversal algorithm and the termination condition for ending the pattern search. Then, we describe a method for calculating tighter local and lattice upper bounds using the new proposed traversal for efficiently pruning uninteresting patterns.

4.1 BDNMiner: A Bi-Directional approach for mining NWC patterns

The main idea behind BDNMiner is using both the cross-K function and $minsupp$ thresholds simultaneously for pruning. To allow pruning based on the cross-K function threshold (i.e. using the lattice and local upper bounds) the algorithm has to perform a top-down traversal of the lattice nodes as previously illustrated in the MTNMiner algorithm. On the other hand, the support measure has a well-known anti-monotone property [2] which indicates that the support of a pattern is always smaller than or equal to the support of its subset patterns. Hence, if the support of a given pattern does not pass the $minsupp$ threshold, all its superset patterns can be pruned since they will never pass the threshold either. Therefore, pruning based on the $minsupp$ threshold requires a bottom-up traversal of the lattice moving from leaf nodes toward the root node. Thus, our proposed BDNMiner algorithm uses a hybrid bi-directional traversal where one level of the lattice is enumerated from each direction at a time. In the bottom-up traversal, nodes are enumerated and can be pruned using the $minsupp$ threshold. In that case, all the parent nodes of the pruned node are also marked as "pruned using $minsupp$". The top-down traversal starts from the top-level and prunes nodes using the lattice and local upper bounds as illustrated in MTNMiner. In Section 4.2, we will also show how this bi-directional traversal strategy can be used to compute tighter local and lattice upper bounds where the pattern cardinalities computed during the bottom-up traversal can be used in calculating the upper bounds during the top-down traversal.

The lattice data structure used by BDNMiner can be shown in Fig. 8a. Unlike MTNMiner, the links between the nodes in this lattice are bi-directional links. Thus, each node has pointers to its parent nodes as well as its children nodes. Additionally, each node has a pointer to its next sibling node in the same lattice to allow the enumeration of a whole level at a time. An array is created with length equal to the number of levels in the lattice where each location in the array stores a pointer to the first node in the corresponding level in the lattice. In addition, each node maintains the following information: (1)$supersetCount$: the maximum cardinality found so far of a superset pattern of this node; and (2)$isPruned$: an integer to indicate if the node was pruned by using the $minsupp$ threshold (=1), or the lattice upper bound (=2), or both (=3) or is unpruned yet (i.e. default value=0).

Algorithm 2 BDNMiner

```
1: enumeratedPatterns ← {}
2: startingEdgeIndex ← CreateStartingEdgeIndexFromMETs
3: lattice ← Create and initialize lattice
4: for each window w=<t_i, t_j>in W_N do                ▷ iterate through all non-compliant windows
5:     for t := δ to 0 do                                ▷ iterate from 0 to the max lag δ preceding w
6:         latticeCp ← CreateDeepCopy(lattice)
7:         nextBottomLevel ← 0
8:         nextTopLevel ← latticeCp.dimensionsNum − 1    ▷ enumerate the leaf-node level
9:         LeafJoinSetCount ← EnumerateOneVariableNodes(latticeCp,enumeratedPatterns)
10:        if all one variable nodes have |C|/T ≤ minsupp then continue
11:        else
12:            nextBottomLevel ← nextBottomLevel+1
13:        enumerateRootNode(latticeCp,latticeCp.root,w,t,δ,nextTopLevel)
14:        isAllLevelPrunedByMinsupp ← false
15:        while (!isAllLevelPrunedByMinsupp) and (nextBottomLevel ≤ nextTopLevel) do
16:            for each node in nextTopLevel do            ▷ enumerate one level from top
17:                enumerateWithUpperBoundPruning(latticeCp,node,w,t,δ)
18:            nextTopLevel ← nextTopLevel-1
19:            if nextBottomLevel ≤ nextTopLevel then   ▷ top-down and bottom-up traversals did not meet
20:                isAllLevelPrunedByMinsupp ← true
21:                for each node in nextBottomLevel do     ▷ enumerate one level from top
22:                    isPruned ← enumerateWithMinsuppPruning(latticeCp,node,w,t,δ,)
23:                    if not isPruned then
24:                        isAllLevelPrunedByMinsupp ← false
25:                nextBottomLevel ← nextBottomLevel+1

26: function enumerateWithUpperBoundPruning(lattice,n,w,t,δ)
27:     if n is marked as pruned by UB_lattice then return
28:     if n is marked as pruned by minsupp then
29:         for each non-leaf child node ch of n not marked as pruned by UB_lattice do
30:             ch.supersetCount ← max(ch.supersetcount,n.supersetcount)
31:         return
32:     if UB_lattice(K̂_{n.C,W_N}(δ)) ≤ ϵ then PruneAllNodeSubsets(n,lattice)
33:     else if UB_local(K̂_{n.C,W_N}(δ)) ≤ ϵ then
34:         for each non-leaf child node ch of n not marked as pruned by UB_lattice do
35:             ch.supersetCount ← max(ch.supersetcount,n.supersetcount)
36:     else                                                ▷ no pruning occurred
37:         C ← expandPattern(n)
38:         if C not in enumeratedPatterns then
39:             [|C|, |C ⋈^δ W_N|] ← Calculate cardinalities using startingEdgeIndex
40:             enumeratedPatterns.put(C)
41:             if |C|/T ≥ minsupp and K̂_{C,W_N}(δ) >ϵ then Output C.
42:             for each non-leaf child node ch of n not marked as pruned by UB_lattice do
43:                 ch.supersetCount ← max(ch.supersetcount,|C|)
44:         else PruneAllNodeSubsets(n)

45: function enumerateWithMinsuppPruning(lattice,n,w,t,δ)
46:     if n is marked as pruned by minsupp then return true
47:     if n is marked as pruned by UB_lattice then return false
48:     C ← expandPattern(n)
49:     if C not in enumeratedPatterns then
50:         [|C|, |C ⋈^δ W_N|] ← Calculate cardinalities using startingEdgeIndex
51:         enumeratedPatterns.put(C)
52:         if |C|/T ≥ minsupp then
53:             if K̂_{C,W_N}(δ) >ϵ then Output C.
54:         else PruneAllNodeSupersets(n) and return true
55:     else▷ C already enumerated ▷ use pattern support to check if its supersets can be pruned
56:         if |C|/T ≥ minsupp then PruneAllNodeSupersets(n) and return true
57:     return false
```

Algorithm 2 shows the pseudo code of BDNMiner. Lines 1-6 are very similar to MTN-Miner where the algorithm starts by creating a lattice for the patterns starting at t time points preceding each non-compliant window, where $0 \leq t \leq \delta$. Line 7 initializes the variable $nextBottomLevel$ to zero to indicate the next level to be enumerated by the bottom-up traversal (i.e. leaf-node level). Similarly, line 8 initializes the variable $nextTopLevel$ to the root node level which is the next level to be enumerated by the top-down traversal. Next, the pattern enumeration step is performed in three phases (lines 9-25). **In phase 1 (lines 9-12)**, patterns represented by all the leaf nodes are enumerated by calculating their cardinalities, join set cardinalities and cross-K function(line 9). The join set cardinalities are also stored in the $LeafJoinSetCount$ array. If the support of a pattern does not exceed the $minsupp$ threshold, all its supserset patterns are marked as "pruned using $minsupp$". Line 10 terminates the search in the current window if all leaf nodes were pruned using $minsupp$. Otherwise, the $nextBottomLevel$ variable is incremented to point to the next level (line 12). **In phase 2 (line 13)**, the root node is enumerated by calculating its cardinality and join set cardinality and evaluating its cross-K function if the $minsupp$ threshold is passed. The join set cardinality of the root is then propagated to the $supersetCount$ variable of all its child nodes and the $nextTopLevel$ variable is decremented. **In phase 3 (lines 14-25)**, the algorithm continues enumerating one level of nodes from each direction at a time. The top-down enumeration is illustrated in lines 16 to 18 by calling the function $EnumerateWithUpperBoundPruning(.)$ for each enumerated node. The bottom-up enumeration occurs in lines 20-25 by calling the function $EnumerateWithMinsuppPruning(.)$. The algorithm continues until the two traversals cross each other (i.e., meet at the same level) or until a complete level is marked as "pruned using $minsupp$" during the bottom-up traversal (line 15). The correctness of this termination condition will be shown in Lemma 4 below.

In the $EnumerateWithUpperBoundPruning(.)$ function (lines 26-44), if the node is already marked as pruned by the lattice upper bound (line 27), the function terminates since all its child nodes should already be marked as pruned. However, if it was marked as pruned by $minsupp$, its $supersetCount$ is propagated to its child nodes similar to MTNMiner (lines 28-31). Otherwise, the node is unpruned. Hence, its lattice and local upper bounds are calculated and dealt with in a similar way as in MTNMiner (lines 32-44).

In the $EnumerateWithMinsuppPruning(.)$ function (lines 45-57), the enumeration terminates immediately if the node was marked as pruned either by $minsupp$ or the lattice upper bound. Otherwise, the pattern is expanded by retrieving it from the input time series. If the pattern was previously enumerated (lines 55-57), its cardinality is retrieved. Then, if the pattern's support is $<minsupp$, all its superset patterns (i.e. all parent nodes) are marked as pruned using $minsupp$. On the other hand, if the pattern was not previously enumerated, its cardinality and join set cardinality are calculated using the $startingEdge$ index. Once again, if the pattern's support is $<minsupp$, all its superset patterns are marked as pruned using $minsupp$. Otherwise, if its cross-K function exceeded ϵ, the pattern is output to the user.

Definition 9 (**Termination condition:**) In BDNMiner, the bi-directional search in a given window (i.e. lattice) terminates when any or both of the following two conditions hold:

(1) $nextBottomLevel > nextTopLevel$

(2) A complete level has been marked as "pruned using $minsupp$" during the bottom-up traversal

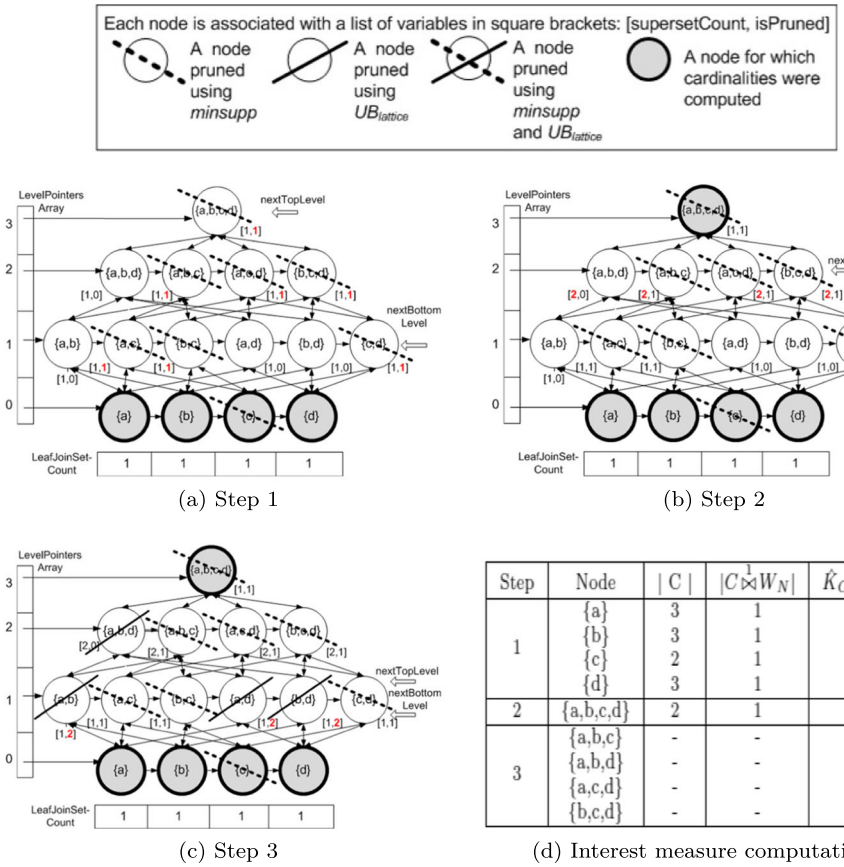
Lemma 4 *The termination condition in Definition 9 is correct.*

Proof To prove Lemma 4, we need to prove that for a lattice representing the patterns in a given window, all patterns must have already been visited or pruned if the termination condition holds. Consider each of the following cases in which the termination condition holds: **Case 1:** $nextBottomLevel > nextTopLevel$. In this case, the next level to be enumerated by the bottom-up traversal is higher than the next level to be enumerated by the top-down traversal. Therefore, each level in the lattice has been already visited by at least one of the two traversals. Hence, all nodes in the lattice have already been visited (A). **Case 2: A complete level has been marked as "pruned using $minsupp$" during the bottom-up traversal**. Due to the anti-monotone property of the support measure, if a pattern's support is less than $minsupp$, all its superset patterns' support must also be less than $minsupp$. Therefore, if a complete level was marked as "pruned using $minsupp$" during the bottom-up traversal, this implies that all their superset patterns are also pruned (see lines 54 and 56 in Algorithm 2). Therefore, all the lattice levels above the pruned level are also pruned. In addition, since the bottom-up traversal already pruned this level, then all levels below it have already been visited by the bottom-up traversal. Therefore, all nodes in the lattice have already been visited or pruned (B). Therefore, from (A) and (B), the termination condition is correct. □

Execution trace Figure 8 shows an example run of BDNMiner for the input data shown in Fig. 7. The MET is of length T=12 and has two non-compliant windows of length L = 3 sec, namely <1,3>and <7,9>. The cross-K function threshold is set to $\epsilon = 3.5$, $\delta = 1$ sec and $minsupp = 1/4$ (for illustration). Again, for brevity, the execution trace shows only the enumeration of candidates within one window <0,2>, which started 1 sec before the non-compliant window <1,3>. Figure 8a shows the lattice created for window <0,2>after enumeration of the leaf-nodes. Their join set cardinalities are shown in the array $LeafJoinSetCount$. Since the pattern of node $\{c\}$ has support = $\frac{2}{12} < minsupp$, node $\{c\}$ and all its parent nodes are marked as "pruned using $minsupp$". The $nextBottomLevel$ variable is also incremented to point to the next higher level. Next, Fig. 8b shows the lattice after enumerating the root node by calculating its cardinality and join set cardinality. The root node is enumerated in order to allow the propagation of its cardinality (=2) to the $supersetCount$ variables of its child nodes. The $nextTopLevel$ variable is also decremented to point to the next lower level. Then, the algorithm starts enumerating one level from each direction until the termination condition is reached. Figure 8c shows the lattice after the top-down traversal enumerates the $nextTopLevel$. Since node $\{a, b, d\}$ is the only remaining unpruned node in this level, its lattice upper bound is calculated. Since that bound is equal to = $\frac{12}{2} \times \frac{\max\{1,1,1\}}{2} = 3 < \epsilon$, the node is pruned and its three child nodes are also marked as pruned (from Theorem 2 and Lemma 3). Next, the $nextTopLevel$ variable is again decremented and the enumeration continues using the bottom-up traversal.

Time	0	1	2	3	4	5	6	7	8	9	10	11
var_a	a_1	a_2	a_3	a_2	a_1	a_2	a_3	a_2	a_1	a_2	a_3	a_2
var_b	b_1	b_1	b_2	b_3	b_1	b_1	b_2	b_2	b_1	b_1	b_1	b_2
var_c	c_1	c_2	c_1	c_1	c_1	c_2	c_1	c_1	c_2	c_2	c_2	c_2
var_d	d_2	d_2	d_3	d_3	d_2	d_2	d_3	d_2	d_1	d_2	d_2	d_3

Fig. 7 Input with two non-compliant windows (best in color)



(d) Interest measure computations

Step	Node	| C |	$\lvert C \overset{1}{\bowtie} W_N \rvert$	$\hat{K}_{C,W_N}(1)$
1	{a}	3	1	2
	{b}	3	1	2
	{c}	2	1	-
	{d}	3	1	2
2	{a,b,c,d}	2	1	-
3	{a,b,c}	-	-	-
	{a,b,d}	-	-	-
	{a,c,d}	-	-	-
	{b,c,d}	-	-	-

Fig. 8 BDNMiner Execution trace (Best viewed in color)

The algorithm terminates when $nextBottomLevel$ exceeds $nextTopLevel$ in the following step.

4.2 Tightening the local and lattice upper bounds

For a given NWC pattern C=$\{S_i(u_i) \mid u_i \in U, U \subseteq V$ and $1 \leq i \leq |U|\}$ and a time lag δ, the local upper bound (see Definition 7) and lattice upper bound (see Definition 8) were defined using an upper bound on $|C \overset{\delta}{\bowtie} W_N|$, namely $Upper_{Loc}(|C \overset{\delta}{\bowtie} W_N|)$ and $Upper_{Lat}(|C \overset{\delta}{\bowtie} W_N|)$ respectively. These $Upper_{Loc}(|C \overset{\delta}{\bowtie} W_N|)$ and $Upper_{Lat}(|C \overset{\delta}{\bowtie} W_N|)$ bounds were calculated using the minimum and maximum join set cardinality, respectively, of all subset patterns of C that consist of only **one** event-sequence (i.e. **one-variable subset patterns**). Hence, in our preliminary MTNMiner algorithm, all leaf nodes are enumerated first and their join set cardinalities are computed. These cardinalities are then used for calculating the local and lattice upper bounds for all the nodes in the lattice.

In our proposed BDNMiner approach, the local and lattice upper bounds calculated in the top-down traversal can be further tightened by using the join set cardinalities of the nodes at the last level enumerated by the bottom-up traversal rather than only the leaf-node level.

For instance, if the last level enumerated by the bottom-up traversal is the level of two-variable nodes (i.e. level 1), the local and lattice upper bounds for node $\{a, b, c\}$ for instance can be calculated using the minimum and maximum join set cardinalities, respectively, of all two-variable subset patterns of node $\{a, b, c\}$ (i.e., patterns of nodes $\{a, b\}$, $\{a, c\}$, and $\{b, c\}$.

To allow the above modification, the bottom-up traversal stores the join set cardinalities of its enumerated nodes in a hash map, $JoinSetCountMap$. The key to this hash map is the node label, e.g., $\{a, b\}$ and the value is the computed join set cardinality of the node. Note that if a node was marked as pruned (i.e using the lattice upper bound), its join set cardinality will not be computed during the bottom-up traversal and hence will be missing from the $JoinSetCountMap$.

Algorithm 3 Computing $Upper_{TiLoc}(|C \overset{\delta}{\bowtie} W_N|)$ for a node

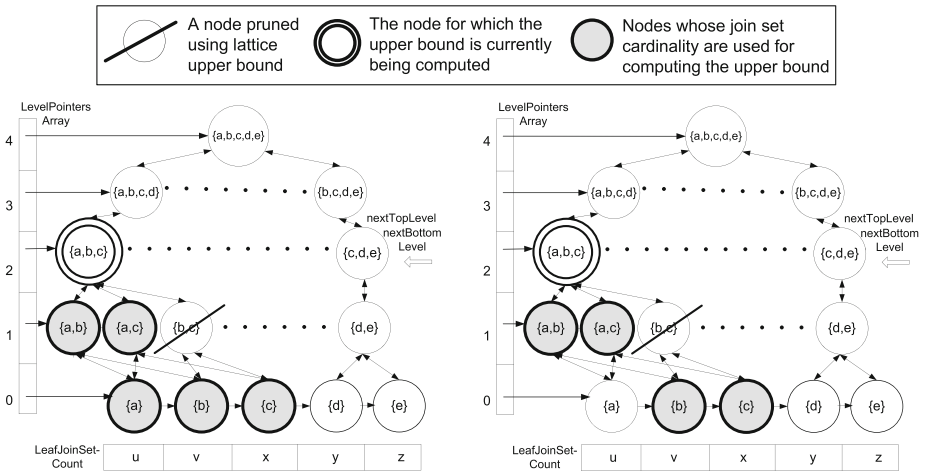
```
1: function UpperTiLoc(node,LastBottomLevel,JoinSetCountMap)
2:     subsetLabels ← Create all node labels representing subset patterns of node whose dimensionality equals lastBottomLevel+1
3:     minJoinSetCount ← Find minimum join set cardinality of all one-variable subsets of node
4:     for each lbl in subsetLabels do
5:         if JoinSetCountMap(lbl) ≠ null then
6:             minJoinSetCount ← MIN(JoinSetCountMap(lbl),minJoinSetCount)
7:     return minJoinSetCount
```

Now, we formally define the tightened local and lattice upper bounds and prove their correctness as shown below.

Definition 10 Tightened local upper bound: For the NWC pattern C, the **tightened local upper bound** of $\hat{K}_{C,W_N}(\delta)$, denoted as $UB_{TightLocal}(\hat{K}_{C,W_N}(\delta))$, can be computed as follows:

$$\begin{aligned} UB_{TightLocal}(\hat{K}_{C,W_N}(\delta)) &= \frac{T}{|W_N|} \times \frac{Upper_{TiLoc}(|C \overset{\delta}{\bowtie} W_N|)}{Lower(|C|)} \\ &= \frac{T}{|W_N|} \times \frac{Upper_{TiLoc}(|C \overset{\delta}{\bowtie} W_N|)}{|\text{superset(C)}|} \end{aligned} \tag{5}$$

The tightened local upper bound for a pattern C is similar to the definition of the local upper bound (Definition 7) except for the part of the numerator representing the upper bound of $|C \overset{\delta}{\bowtie} W_N|$, namely $Upper_{TiLoc}(|C \overset{\delta}{\bowtie} W_N|)$, which is calculated based on Algorithm 3. The algorithm takes three inputs: the node being enumerated (i.e. node of pattern C), the index of the last level enumerated by the bottom-up traversal, and the $JoinSetCountMap$ storing the join set cardinalities of enumerated nodes. As shown in the algorithm, $Upper_{TiLoc}(|C \overset{\delta}{\bowtie} W_N|)$ is calculated using the minimum of the join set cardinalities of all one-variable subset patterns of C and the join set cardinalities of any subset pattern of C that lies in the last level enumerated by the bottom-up traversal (i.e. patterns whose dimensionality equals the number of variables in the $lastBottomLevel$). Figure 9a illustrates the nodes used for computing the $Upper_{TiLoc}(|C \overset{\delta}{\bowtie} W_N|)$ for node {a,b,c} assuming that level 1 (i.e. the level of two-variable nodes) is the last level enumerated from the bottom of the lattice and that the node {b,c} has been marked as pruned. Thus, the grey shaded nodes are the nodes whose join set cardinalities are used in the computation, and the minimum of their join set cardinalities is returned as the value for $Upper_{TiLoc}(|C \overset{\delta}{\bowtie} W_N|)$.



(a) Nodes used for computing $Upper_{TiLoc}(|C \overset{\delta}{\bowtie} W_N|)$ for node {a,b,c}

(b) Nodes used for computing $Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|)$ for node {a,b,c}

Fig. 9 An example illustrating nodes used for computing the tightened upper bounds for node {a,b,c})

Since node {b,c} was marked as pruned, its join set cardinality is not used. Next, we show that the tightened local upper bound is a correct upper bound of the cross-K function interest measure.

Lemma 5 *Given an NWC pattern C and a time lag δ, $Upper_{TiLoc}(|C \overset{\delta}{\bowtie} W_N|)$ is an upper bound of $|C \overset{\delta}{\bowtie} W_N|$.*

Proof *To prove this lemma, we need to prove that for a pattern C, Algorithm 3 returns an upper bound of $|C \overset{\delta}{\bowtie} W_N|$. Algorithm 3 starts by computing the minimum join set cardinality of all one-variable subsets of C, namely $\{S_i\}$, where $\{S_i\} \subseteq C$, $1 \leq i \leq Dim(C)$ and stores it in the variable $minJoinSetCount$ (line 3). Since all subsets of C have a join set cardinality at least equal to that of C, we now have $minJoinSetCount \geq |C \overset{\delta}{\bowtie} W_N|$. Now, let $lastBottomLevel$ be the the last level enumerated by the bottom-up traversal. From Algorithm 3, we know that the value of $minJoinSetCount$ only changes if a subset pattern of C lying at $lastBottomLevel$ was enumerated and its the join set cardinality is smaller than the current value of $minJoinSetCount$ (lines 4 to 6). Again, since all subset patterns of C have a join set cardinality at least equal to that of C, therefore $minJoinSetCount$ (i.e. the returned value) will always remain greater than or equal to $|C \overset{\delta}{\bowtie} W_N|$. Thus, Algorithm 3 will always return an upper bound of $|C \overset{\delta}{\bowtie} W_N|$*

Theorem 3 *Given an NWC pattern C and a time lag δ, $UB_{TightLocal}(\hat{K}_{C,W_N}(\delta))$ is an upper bound of $\hat{K}_{C,W_N}(\delta)$.*

Proof From Theorem 1, we already know that UB_{local} is an upper bound of $\hat{K}_{C,W_N}(\delta)$. Now, the $UB_{TightLocal}$ definition is similar to the UB_{local} definition except for replacing the

$Upper_{Loc}(|C \overset{\delta}{\bowtie} W_N|)$ term by the $Upper_{TiLoc}(|C \overset{\delta}{\bowtie} W_N|)$ term for upper bounding $|C \overset{\delta}{\bowtie} W_N|$. Since from Lemma 5, we know that $Upper_{TiLoc}(|C \overset{\delta}{\bowtie} W_N|)$ is also an upper bound of $|C \overset{\delta}{\bowtie} W_N|$, then $UB_{TightLocal}(\hat{K}_{C,W_N}(\delta))$ is also an upper bound of $\hat{K}_{C,W_N}(\delta)$. □

Definition 11 Tightened lattice upper bound: For the NWC pattern C, the **tightened lattice upper bound** of $\hat{K}_{C,W_N}(\delta)$, denoted as $UB_{TightLattice}(\hat{K}_{C,W_N}(\delta))$, can be computed as follows:

$$UB_{TightLattice}(\hat{K}_{C,W_N}(\delta)) = \frac{T}{|W_N|} \times \frac{Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|)}{Lower(|C|)} = \frac{T}{|W_N|} \times \frac{Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|)}{|\text{superset(C)}|} \quad (6)$$

The tightened lattice upper bound for a pattern C is also similar to the definition of the lattice upper bound (Definition 8) except for the part of the numerator representing the upper bound of $|C \overset{\delta}{\bowtie} W_N|$, namely $Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|)$, which is calculated based on Algorithm 4. In this algorithm, the bound is calculated using the maximum of the join set cardinalities of all subset patterns of C that lie in the last level enumerated from the bottom of the lattice. If any of these subset patterns does not exist in the $JoinSetCountMap$, all the one-variable patterns of this subset are used instead. This is performed to maintain the pruning power of the lattice upper bound (i.e., the ability to prune all node descendants if the node's lattice upper bound does not exceed the cross-K function threshold) as will be proved in Lemma 6 below. Figure 9b shows an example for computing $Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|)$ for node {a,b,c} where the shaded nodes are the nodes whose join set cardinalities are retrieved, and their maximum is returned. Since node {b,c} has been marked as pruned, its join set cardinality cannot be found in the $JoinSetCountMap$ and thus the join set cardinalities of its one-variable subset patterns, namely {b} and {c} are used instead.

Algorithm 4 Computing $Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|)$ for a node

```
1: function UpperTiLat(node,LastBottomLevel,JoinSetCountMap)
2:     subsetLabels ← Create all node labels representing subset patterns of node whose dimensionality equals lastBottomLevel+1
3:     maxJoinSetCount ← -∞                                   ▷ very small value
4:     for each lbl in subsetLabels do
5:         if JoinSetCountMap(lbl) ≠ null then
6:             maxJoinSetCount ← MAX(JoinSetCountMap(lbl),maxJoinSetCount)
7:         else
8:             maxLeafJoinSetCount ← Find maximum join set cardinality of all one-variable subset nodes of lbl
9:             maxJoinSetCount ← MAX(maxLeafJoinSetCount,JoinSetCountMap(lbl))
10:    return maxJoinSetCount
```

Next, we show that the tightened lattice upper bound is a correct upper bound of the cross-K function interest measure.

Theorem 4 *Given an NWC pattern C and a time lag δ, $UB_{TightLattice}(\hat{K}_{C,W_N}(\delta))$ is an upper bound of $\hat{K}_{C,W_N}(\delta)$.*

Proof From Theorem 2, we already know that $UB_{lattice}$ is an upper bound of $\hat{K}_{C,W_N}(\delta)$. The $UB_{TightLattice}$ definition is similar to the $UB_{lattice}$ definition except for replacing the $Upper_{Lat}(|C \overset{\delta}{\bowtie} W_N|)$ by the $Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|)$ term. Therefore, to prove this theorem, we only need to prove that $Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|)$ is also an upper bound of $|C \overset{\delta}{\bowtie} W_N|$. Thus, we need to prove that for a pattern C, Algorithm 4 returns an upper bound of $|C \overset{\delta}{\bowtie} W_N|$. Now, let $lastBottomLevel$ be the the last level enumerated by the bottom-up traversal. Algorithm 4 sets the variable $maxJoinSetCount$ to the maximum join set cardinality of the subset patterns of C lying at $lastBottomLevel$ (line 6). If a subset of C at $lastBottomLevel$ has been pruned and thus its join set cardinality was not found in $JoinSetCountMap$, the maximum join set cardinality of its one-variable subsets are used instead (line 8). Since all subsets of C (including one-variable subsets and subsets lying at $lastBottomLevel$) have a join set cardinality at least equal to that of C, therefore $maxJoinSetCount$ (i.e. return value) is always greater than or equal to $|C \overset{\delta}{\bowtie} W_N|$. Thus, Algorithm 4 will always return an upper bound of $|C \overset{\delta}{\bowtie} W_N|$. □

In the following lemma, we prove that the tightened lattice upper bound has a conditional upper bounding property which allows pruning all descendants of a node whose tightened lattice upper bound does not exceed the cross-K function threshold provided that the descendant node lies at a level above the last level enumerated by the bottom-up traversal.

Lemma 6 ***Conditional Upper Bounding Property for the Tightened Lattice Upper Bound:*** *Given an NWC pattern C and a time lag δ, then for any pattern C' ⊂ C, lying above the last level enumerated by the bottom-up traversal, $UB_{TightLattice}(\hat{K}_{C,W_N}(\delta))$ is also an upper bound of $\hat{K}_{C',W_N}(\delta)$.*

Proof Let $lastBottomLevel$ be the last level enumerated by the bottom-up traversal. Let $SubsetLabels_C$ be the set of all node labels representing subset patterns of C lying at $lastBottomLevel$. Based on Algorithm 4, $Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|)$ is set to the maximum of the join set cardinalities of nodes in $SubsetLabels_C$ which have not been pruned and the the join set cardinalities of one-variable subsets of the nodes in $SubsetLabels_C$ which have been pruned. Since for every node x ∈ $SubsetLabels_C$, the join set cardinality of all one-variable subsets of x is greater than or equal to the join set cardinality of x, therefore we have $Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|) \geq \max_{x \in SubsetLabels_C}(|x.pattern \overset{\delta}{\bowtie} W_N|)$.(A)

Now for any pattern C' ⊂ C, lying above $lastBottomLevel$, let $SubsetLabels_{C'}$ be the set of all node labels representing subset patterns of C' lying at $lastBottomLevel$. Since C' ⊂ C, then for every y ∈ $SubsetLabels_{C'}$, y also belongs to $SubsetLabels_C$.(B)

From (A) and (B), we get $Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|) \geq \max_{x \in SubsetLabels_C}(|x.pattern \overset{\delta}{\bowtie} W_N|) \geq \max_{y \in SubsetLabels_{C'}}(|y.pattern \overset{\delta}{\bowtie} W_N|)$.(C)

In addition, for every y in $SubsetLabels_{C'}$, we have $|y.pattern \overset{\delta}{\bowtie} W_N|) \geq |C' \overset{\delta}{\bowtie} W_N|$ since y is a subset of C'.(D)

Therefore, from (C) and (D), we get $Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|) \geq \max_{x \in SubsetLabels_C}(|x.pattern \overset{\delta}{\bowtie} W_N|) \geq \max_{y \in SubsetLabels_{C'}}(|y.pattern \overset{\delta}{\bowtie} W_N|) \geq |C' \overset{\delta}{\bowtie} W_N|$ (E).

In addition, from Definition 11, $Lower(|C|) = |superset(C)| \leq |C'|$.(F)
Therefore, from (E) and (F), we get $UB_{TightLattice}(\hat{K}_{C,W_N}(\delta)) = \frac{T}{|W_N|} \times \frac{Upper_{TiLat}(|C \overset{\delta}{\bowtie} W_N|)}{Lower(|C|)} \geq \frac{T}{|W_N|} \times \frac{|C' \overset{\delta}{\bowtie} W_N|}{|C'|} = \hat{K}_{C',W_N}(\delta)$. □

5 Theoretical evaluation

In this section, we formally prove the correctness and completeness of our proposed BDNMiner algorithm.

5.1 Correctness of BDNMiner

Lemma 7 *The BDNMiner algorithm is correct. Correctness means that every NWC pattern C discovered by the algorithm has* $\hat{K}_{C,W_N}(\delta) > \epsilon$ *and support* $\geq minsupp$.

Proof According to Algorithm 2, the BDNMiner algorithm only outputs an NWC pattern in any of the following lines: line 9, line 13, line 41 and lines 52-53. In lines 9 and 13, the EnumerateOneVariableNodes() and EnumerateRootNode() functions, respectively, evaluate the pattern support and cross-K function interest measure and the pattern is output only if its interest measure $\hat{K}_{C,W_N}(\delta)$ exceeds ϵ and its support is at least equal to the $minsupp$ threshold. Similarly, lines 41 and 52-53 explicitly ensure that the pattern cross-K function exceeds ϵ and the pattern support is at least equal to the $minsupp$ threshold. Therefore, the BDNMiner algorithm is correct. □

5.2 Completeness of BDNMiner

Lemma 8 *The BDNMiner algorithm is complete. Completeness means that every NWC pattern C with* $\hat{K}_{C,W_N}(\delta) > \epsilon$ *and support* $\geq minsupp$ *is reported by BDNMiner.*

Proof According to Algorithm 2 (lines 3-6), BDNMiner creates lattice nodes for all patterns that start within time t from all non-compliant windows, where $0 \leq t \leq \delta$. Hence, according to Definition 5, all possible NWC patterns have a corresponding node in the created lattices. Then, the BDNMiner algorithm performs a bi-directional traversal where one complete level is enumerated from each direction at a time until the termination condition is satisfied (lines 7-25). Hence, since the termination condition is correct (from Lemma 4), all created nodes are either visited for enumeration or pruned. Therefore, for the algorithm to be complete, we only need to prove that no pruning occurs for any node whose pattern has $\hat{K}_{C,W_N}(\delta) > \epsilon$ and support $\geq minsupp$. In Algorithm 2, a node n representing an NWC pattern C is pruned only in one of the following six cases: **Case 1: (lines 32 and 33)** its lattice upper bound $\leq \epsilon$ (line 32) or local upper bound $\leq \epsilon$ (line 33). In this case, according to Theorems 2 and 1 respectively (or Theorems 4 and 3 if the tightened lattice and local upper bounds were used), $\hat{K}_{C,W_N}(\delta) \leq \epsilon$ and n can be pruned. **Case 2: (also in line 32)** the lattice upper bound of one of its ancestor nodes $\leq \epsilon$. From Lemma 3, if the lattice upper bound of one of the ancestors of n $\leq \epsilon$, then the lattice upper bound of n $\leq \epsilon$, and hence its $\hat{K}_{n.C,W_N}(\delta)$ is also $\leq \epsilon$ if the $supersetCount$ is kept monotonically increasing. So, now we only need to prove that $supsersetCount$ monotonically increases as we go down the lattice and the dimensionality of the patterns decrease. According to Algorithm 2, the

$supersetCount$ of a node is either kept the same or is set to a larger value from one of its parent or ancestor nodes (lines 30, 35 and 43). Hence, $supersetCount$ never decreases as we go down the lattice and thus the condition holds. Also, if the tightened lattice upper bound was used in line 32, from Lemma 6, the tightened lattice upper bound of a node is also an upper bound on all its descendant nodes that are not already enumerated by the bottom-up traversal. Hence, if the bound of an ancestor node of n is $\leq \epsilon$, then the cross-K function of the descendant node n is also $\leq \epsilon$ and n can be pruned. A node n can also be pruned by: **Case 3: (lines 44 and 55)** the pattern was already in the $enumeratedPatterns$ table. So, clearly the pattern has been already considered. **Case 4: (also line 44)** one of its ancestor nodes was in the $enumeratedPatterns$ table. This case happens when an ancestor node pattern has already been enumerated before in another window. Hence, all its descendant patterns have already been considered in that window and there is no need to reconsider those patterns in the current window and thus n can be pruned. **Case 5: (line 54)** the pattern support is $<minsupp$ so n can be pruned. **Case 6: (lines 54 and 56)** one of its descendant nodes had a pattern support $<minsupp$. Due to the anti-monotone property of the support measure, if a pattern is infrequent (i.e. has support $<minsupp$), then all its parent nodes are also infrequent [2]. Hence, if a descendant node of n has support $<minsupp$, then the pattern in n must also have a support $<minsupp$ and can be pruned. □

6 Case study

Patterns of non-compliant CO_2 emissions: With ever mounting evidence of human caused global climate change, decreasing the production of heat trapping or "greenhouse" gases (GHG) has become a top priority of researchers around the world [19]. One of the key human produced GHGs which makes up nearly 83 % of all heat trapping gasses is carbon dioxide or CO_2 [28] meaning reduction of CO_2 emissions is pertinent and necessary. CO_2 emissions from engine powered vehicles are directly proportional to the quantity of fuel used; therefore CO_2 emissions reduction must be accomplished through reduced fuel use. Furthermore, with over 250 million vehicles registered in the US [11], a small improvement would have a major impact.

To evaluate the effectiveness of the BDNMiner algorithm in detecting non-compliant emissions behavior, we conducted a case study of engine CO_2 emissions using real-world sensor data collected from on board a transit bus in the Minneapolis-St. Paul area, USA. The dataset measured several engine and environmental variables at a rate of 1 Hz. Data points covered roughly 19 days ($\approx$ 176 trips) on three routes with different average speeds to build a dataset representative of transit bus operation and ensure that the data was not biased by a specific route. The mass-specific CO_2 emissions were calculated from the diesel fueling rate, a reasonable assumption for diesel vehicles [26]. With CO_2 as the focus, the non-compliant windows of CO_2 emissions were defined as windows of length $L = 5$ sec in which the average CO_2 in gm/kW-h exceeded the Environmental Protection Agency (EPA) vocational standard threshold ($avgCO2$) of 800 [29] and the percentage of increase in CO_2 exceeded $PincT = 100$ %. We used a temporal cross-K function threshold $\epsilon = 55$, $\delta = 1$ sec, and $minsupp = 0.001$ %.

The variables tested in this study were the engine crank shaft rotations per minute (Engine RPM), exhaust gas recirculation flow rate (EGR kgph), exhaust gas pressure, wheel speed, engine brake power, GPS elevation change, vehicle acceleration, engine fuel rate

Table 5 Interesting CO_2-related NWC patterns (smaller indexes indicate smaller values)

ID	NWC Pattern C	$\hat{K}_{C,W_N}(1)$	support(C)
1	Engine RPM: $\{s_3\ s_3\ s_3\ s_2\ \mathbf{s_{22}}\}$ Wheel speed: $\{w_0 w_0 w_0 w_0 w_0\}$ EGR kgph: $\{g_0 g_0 g_0 g_0 g_0\}$	62.64	0.0021%
2	Engine RPM: $\{\mathbf{s_4}\ s_2\ s_2\ s_2\ s_2\}$ Wheel speed: $\{\mathbf{w_3}\ w_2\ w_2\ w_2\ w_1\}$	62.64	0.0011%
3	Fuel rate: $\{f_0\ f_1\ f_2\ f_3\ \mathbf{f_5}\}$ Brake Power: $\{p_{16}\ p_{16}\ p_{17}\ p_{17}\ p_{17}\}$ EGR kgph: $\{g_0 g_0 g_0 g_0 g_0\}$	62.64	0.0011%
4	Engine RPM: $\{\mathbf{s_4}\ s_3\ s_2\ s_2\ s_2\}$ Wheel speed: $\{\mathbf{w_3}\ w_3\ w_2\ w_2\ w_1\}$ Brake Power: $\{p_{16}\ p_{16}\ p_{17}\ p_{17}\ p_{17}\}$	62.64	0.0011%
5	Wheel speed: $\{\mathbf{w_3}\ w_3\ w_2\ w_2\ w_1\}$ Fuel rate: $\{f_0\ f_1\ f_2\ f_3\ \mathbf{f_4}\}$ Brake Power: $\{p_{16}\ p_{16}\ p_{17}\ p_{17}\ p_{17}\}$	62.64	0.0011%
6	Fuel rate: $\{f_5\ f_6\ f_5\ f_7\ \mathbf{f_{14}}\}$ Acceleration: $\{a_{34}\ a_{34}\ a_{34}\ a_{34}\ a_{35}\}$	62.64	0.0013%
7	Wheel speed: $\{w_0 w_0 w_0 w_0 w_0\}$ Fuel rate: $\{f_6\ f_6\ f_6\ f_6\ \mathbf{f_{15}}\}$ Acceleration: $\{a_{34}\ a_{34}\ a_{34}\ a_{34}\ a_{35}\}$ Elevation change: $\{e_{100}\ e_{100}\ e_{100}\ e_{100}\ e_{100}\}$ EGR kgph: $\{g_0 g_0 g_0 g_0 g_0\}$	62.64	0.0012%
8	Wheel speed: $\{w_0 w_0 w_0 w_0 w_0\}$ EGR kgph: $\{g_0 g_0 g_0 g_0 g_0\}$ Torque Demand Diff: $\{t_{15}\ t_{15}\ t_{15}\ t_{19}\ \mathbf{t_{23}}\}$	62.64	0.0011%
9	Acceleration: $\{a_{34}\ a_{34}\ a_{34}\ a_{34}\ a_{33}\}$ EGR kgph: $\{g_0 g_0 g_0 g_0 g_0\}$ Torque Demand Diff: $\{\mathbf{t_{18}}\ t_{17}\ t_{16}\ t_{16}\ t_{15}\}$	62.64	0.0013%

and engine torque demand difference (Torque Demand Diff) as these are likely to influence CO_2 production and exhibit change within the specified window length and time lag. Most parameters had equal length windows; however a modified window was necessary for engine RPM because of the disproportionate amount of low and high idle periods with narrow RPM bands. Engine idle operation is essentially the default states for the engine when the vehicle is stopped and no engine power is required. Therefore identifying when idling occurs is necessary for understanding what the engine is doing.

BDNMiner identified 29,510 non-compliant windows and 1,067 patterns. The algorithm distinguished multiple instances of elevated emissions corresponding to the engine reacting to external disturbances such as the driver slowing down or accelerating. Table 5 shows a select number of interesting output patterns with the highest cross-K function values where elevated emissions were detected. Pattern 1 shows an instance where the bus engine RPM transitions from the low idle window to a slightly slower engine speed and then jumps up to a high engine speed all while the vehicle wheel speed is in the 0 to 5 km/h range. This is indicative of the driver shifting the bus into drive and accelerating from a break or end-of-route point. Pattern 2 illustrates a series where a transition from a higher speed to a slower speed causes high emissions, and pattern 3 illustrates instances where change in engine power output also results in elevated emissions. Interestingly, patterns 4 and 5 show two engine signatures where elevated emissions are detected as the vehicle is slowing and

the engine power production is increasing. At initial glance, this behavior is counter intuitive since a slowing vehicle should not need power to decelerate. However, when taking into account the dynamics of vehicle systems, this behavior can be explained by vehicle accessory power consumption (i.e. water pump, alternator and power steering). These accessories require continuous power to operate. As the vehicle slows, decreasing power comes from the wheels, requiring the engine to compensate. This behavior is further explained by the steady increase in fueling in pattern 5.

Patterns 6 and 7 provide two scenarios where elevated emissions co-occur with an abrupt increase in bus fueling rate and acceleration. This scenario is indicative of the driver accelerating, which results in an increase in CO_2 while the engine works to increase its power output. It is possible that acceleration patterns of different drivers would show up as "signatures" in the fueling rates much like the differences in fueling between patterns 6 and 7. Lastly, two interesting, yet contrasting patterns which could help explain the other cases are patterns 8 and 9. Unlike previous patterns, these are examples of the bus reacting to changes in the demand on the vehicle from external sources (e.g., the air conditioning turns on requiring more engine torque). Both of these cases are dependent on the change in torque demand difference which quantifies the discrepancy between the desired and actual engine output. However, pattern 8 depends on increasing demand difference which may be a result of a vehicle accessory turning on (e.g., air conditioning, power steering, heating fans). Contrarily, pattern 9 depends on decreasing demand difference possibly arising from the bus coasting towards a stop. The occurrence of these two patterns indicates that the increased emissions is a result of the vehicle's system adapting to change. This confirms the findings of others [4, 12] and should prompt engine researchers to investigate strategies for optimally handling these specific transient cases. Such improvements would help in the effort to reduce CO_2 emissions.

Patterns of non-compliant NO_x emissions Using the same dataset, we conducted a second case study on NO_x emissions. NO_x is a harmful type of engine emission whose inhalation is detrimental to lung function and increases the health risks for sensitive populations. Additionally, atmospheric NO_x emissions are precursors of the harmful ground level ozone and acid rain formation [27]. In this case study, the non-compliant windows of NO_x emissions were defined as windows of length $L = 5$ sec in which the average NO_x in gm/kW-h exceeded the Environmental Protection Agency (EPA) test threshold ($avgNO_xT$) of 0.267 [8] and the percentage increase in NO_x exceeded $PincT = 100\%$. We used a temporal cross-K function threshold $\epsilon = 15$, $\delta = 2$ sec, and $minsupp = 0.01\%$. The variables used for this case study included engine RPM, engine torque, engine power, wheel speed, and acceleration which can typically influence the increase in NO_x. Additionally, since the production of NO_x is heavily dependent on temperature [26], the engine intake temperature, coolant temperature, and selective catalytic reduction (SCR) system intake temperature were also added to the list of prescribed variables. Except for the coolant temperature variable which is not expected to show a significant change within a window length (L) of 5 sec and a lag time (δ) of 2 sec, the remaining variables selected for this study on NOx emissions are expected to show change within the specified window length and lag parameters. Similar to the CO_2 emission case, most of the variables had equal length intervals; however for the engine RPM, additional modified windows were also created to account for the narrow windows of engine idling.

The number of identified non-compliant windows was 98,290, generating 1,159 NWC patterns. Analysis of the output shows that BDNMiner was able to correctly identify the high NO_x association with low engine load and slow speed driving that was previously found by

Table 6 Interesting NO_x-related NWC patterns (smaller indexes indicate smaller values)

ID	NWC Pattern C	$\hat{K}_{C,W_N}(2)$	support(C)
1	Wheel speed: $\{w_0\ w_0\ w_0\ w_1\ \mathcal{W}_2\}$	21.57	0.66%
2	Engine RPM: $\{s_1\ s_2\ \mathcal{S}_3\ \mathcal{S}_3\ \mathcal{S}_3\}$ Engine power: $\{r_5 r_5 r_5 r_5 r_5\}$ Wheel speed: $\{w_0 w_0 w_0 w_0 w_0\}$ Acceleration: $\{a_{16}\ a_{16}\ a_{17}\ a_{17}\ a_{17}\}$	16.28	0.01006%
3	Engine RPM: $\{s_1\ s_1\ s_2\ \mathcal{S}_3\ \mathcal{S}_3\}$ Engine power: $\{r_5 r_5 r_5 r_5 r_5\}$ Wheel speed: $\{w_1\ w_0 w_0 w_0 w_0\}$	17.15	0.011%

Misra et al. [20]. The association between slow driving speed and high NO_x is shown in the first row of Table 6. This NWC pattern illustrates that accelerating at speeds between 0 and 15 km/h is highly associated with elevated NO_x conditions. The output pattern shown in the second row of Table 6 illustrates the association between high NO_x output and low engine load. In this instance, the wheel speed was between 0 and 5 km/h, and the engine load was around 10% of the rated load as indicated by the low engine power bin which would constitute a low load condition. These findings confirm that NWC pattern discovery can correctly identify patterns associated with high NO_x. A particularly interesting finding is also shown in the last row of Table 6. In this NWC pattern, the wheel speed appears to decrease from the start of the window, while the engine RPM appears to increase substantially, resulting in what can be thought of as a counter intuitive vehicle operation. A potential explanation of this case could be the effect of some factors such as a down-shift in the transmission. Further investigation would be needed to understand the true cause of this finding, but having these windows identified provides engine researchers with a specific starting point and an insight into the parameters involved.

7 Experimental evaluation

- MTNMiner: The multi-parent tracking algorithm discussed in Section 3.3.
- BDNMiner-LO: This is a version of the BDNMiner algorithm where the lattice and local upper bounds are calculated similar to the MTNMiner algorithm (i.e. as proposed in Section 3.2). LO indicates that the bounds are calculated using the join set cardinalities of the **l**eaf nodes **o**nly.
- BDNMiner-LBL: This is the BDNMiner algorithm using the tightened lattice and local upper bounds proposed in Section 4.2. LBL indicates that the bounds are calculated using the join set cardinalities of the **l**ast **b**ottom **l**evel that was enumerated.

7.1 Experimental setup

Experiments were performed using the real-world dataset used in the NO_x case study with a time series of length T=100,000 points. The non-compliant windows were defined as windows of length 5 sec in which the average of NO_x emissions in gm/kW-h exceeded the EPA standard threshold ($avgNO_xT$) of 0.267, and the percentage of increase in NO_x exceeded $PincT$ = 100 %. The default parameter values were: T = 50,000 points, $|V|$ = 8, ϵ = 15, δ = 2 sec, L = 5 sec, $minsupp$ = 0.005 %, $avgNO_xT$ = 0.267 and $PincT$ = 100%, unless stated otherwise. Algorithms were implemented using the Java programming

language. All experiments were run on a machine with an Intel Xeon Quad Core 3.00 GHz processor with 64 GB RAM.

7.2 Experimental results

In this subsection, we focus on evaluating the algorithmic refinements of the BDNMiner algorithm as compared to MTNMiner. However, we also evaluated the effect of the $startingEdge$ index by running two versions of the naive approach in order to separate the effect of the index from all the other pruning filters. The first version of the naive approach uses a linear scan of the data to calculate the cardinality of each pattern and its join set, while the other version uses the $startingEdge$ index. Figure 10 shows the execution times of both versions. As can be seen, the $startingEdge$ index leads to substantial computational savings by reducing the time required for cardinality counting. At T=10^5 points, the naive approach using the $startingeEdge$ index was 8.5 times faster than the linear scan version. Consequently, we used the $startingEdge$ index as the method for cardinality counting in all of our experiments.

Effect of time series length (T) We ran the BDNMiner (both versions) and MTNMiner algorithms on subsets of the dataset with 1000, 10,000, 50,000 and 100,000 points where each subset was a contiguous set of trips. Figure 11a shows the execution times for both algorithms and Fig. 11b shows the corresponding speedup ($=\frac{\texttt{MTNMiner execution time}}{\texttt{BDNMiner-LBL execution time}}$). As can be seen, both versions of the BDNMiner algorithm significantly outperform the MTNMiner algorithm. We can also see that although BDNMiner-LBL uses tighter upper bounds, its execution time is very similar to the BDNMiner-LO algorithm. This can be attributed to the additional overhead in calculating the tightened upper bounds where all subsets of dimensionality equal to the number of variables in the last bottom level enumerated have to be generated for each node. Overall, the computational savings of BDNMiner increase as the length of the time series increases. At 100,000 points, BDNMiner is more than an order of magnitude faster than the MTNMiner.

Effect of the number of variables ($|V|$) Figure 11c shows the execution times for both algorithms as the number of variables increases and Fig. 11d shows the corresponding speedup. While the execution time of the MTNMiner algorithm increases exponentially, the growth of the BDNMiner execution time is much slower due to the use of minimum support pruning. At $|V|$=14 variables, BDNMiner is 48 times faster than MTNMiner. Figure 11e compares the execution times of BDNMiner-LBL and BDNMiner-LO only to clarify their

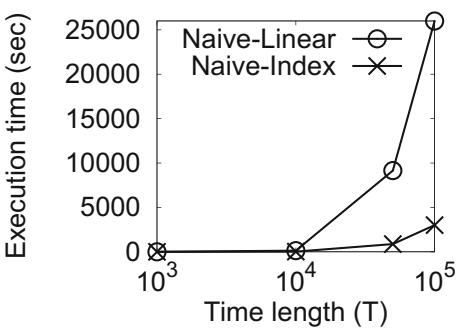
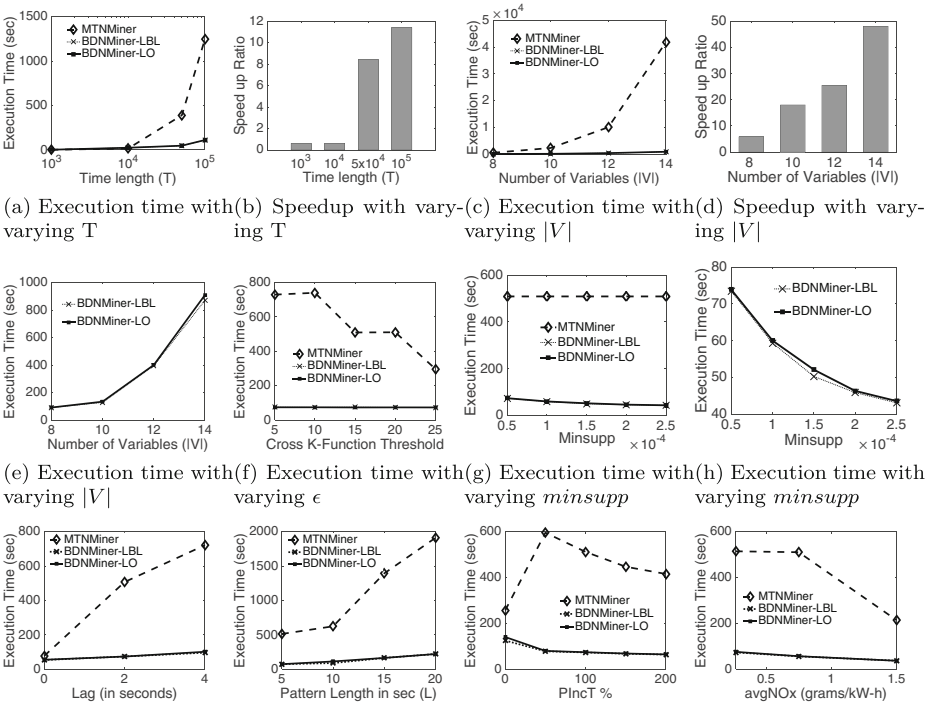


Fig. 10 Execution time with varying T

distinction. Although both algorithms have very similar execution times, BDNMiner-LBL runs slightly faster as the number of variables increases due to using tighter upper bounds.

Effect of temporal cross-K function threshold (ϵ) Figure 11f shows the execution times for MTNMiner and both versions of BDNMiner as the temporal cross-K function threshold ϵ increases. As can be seen, the execution time for MTNMiner significantly decreases with the increase in ϵ. This happens because most of the candidate patterns can be pruned at higher thresholds for the interest measure. On the other hand, the execution times of both versions of BDNMiner only slightly decrease with the increase in ϵ and both versions have the same trend. This indicates that the minimum support pruning also plays a dominant role in BDNMiner's execution time savings.

Effect of minimum pattern support threshold ($minsupp$) Figure 11g shows the execution times of MTNMiner, BDNMiner-LO and BDNMiner-LBL as $minsupp$ increases from 0.005 % to 0.025 %. The increase in the minimum support threshold has no effect on MTNMiner. Its computational cost remains constant since the algorithm only uses $minsupp$ as a post-processing step before a pattern is output to the user. By contrast, the execution time of both BDNMiner versions decreases with the increase in the $minsupp$ threshold since more patterns can be pruned during the bottom-up traversal. Figure 11h compares the execution times of BDNMiner-LBL and BDNMiner-LO only to illustrate the difference in



(a) Execution time with varying T (b) Speedup with varying T (c) Execution time with varying $|V|$ (d) Speedup with varying $|V|$

(e) Execution time with varying $|V|$ (f) Execution time with varying ϵ (g) Execution time with varying $minsupp$ (h) Execution time with varying $minsupp$

(i) Execution time with varying δ (j) Execution time with varying L (k) Execution time with varying $PIncT$ (l) Execution time with varying $avgNO_xT$

Fig. 11 Execution time for BDNMiner vs. MTNMiner

performance. As can be seen, BDNMiner-LBL runs slightly faster than BDNMiner-LO due to the use of the tighter lattice and lower upper bounds.

Effect of time lag (δ) To observe the effect of the maximum time lag between a pattern and a non-compliant window, we measured the execution times with δ varying from 0 to 4 secs. Figure 11i shows that a larger δ value results in an increase in the execution time for both algorithms. The reason is that more time is needed to enumerate the larger number of temporal windows preceding each non-compliant window. Nevertheless, both versions of BDNMiner consistently outperform MTNMiner with computational savings increasing as the value of δ increases.

Effect of pattern length (L) Figure 11j shows the effect of pattern length on the MTNMiner, BDNMiner-LO and BDNMiner-LBL algorithms. As the pattern length increases, the execution time of all algorithms increase due to the increase in the cost of calculating the pattern cardinality. Nevertheless, both versions of BDNMiner always outperform MTNMiner.

Effect of the number of non-compliant windows The effect of the number of non-compliant windows was studied by varying two variables which control the non-compliant window definition, namely, $PIncT$ and $avgNO_xT$. Figure 11k shows the execution times as $PIncT$ increases from 0 % to 200 %. The computational cost of MTNMiner decreases as $PIncT$ was increased from 50 % to 200 %. This is due to the decrease in the number of non-compliant windows, which reduces the total number of patterns enumerated. However, at $PIncT = 0$ %, the number of non-compliant windows was very high, resulting in a large decrease in the cross-K function values for all the candidate patterns. This occurred because the cardinality of non-compliant windows $|W_N|$ lies in the cross-K function's denominator and no output patterns were produced. As a result, most patterns were pruned by MTNMiner in this case, leading to a large reduction in execution time. However, as $PIncT$ increased from 0 % to 50 %, the number of non-compliant windows decreased substantially (from 12,931 to 6,026 windows), leading to higher values of the cross-K function and less pruning. Then, as $PIncT$ increased to 100 %, the number of non-compliant windows exhibited a smaller decrease (from 6,026 to 4,845 windows). At this smaller decrease, the reduction in the overall computation pattern enumeration time was higher than the pruning lost by the increase of the cross-K function values, leading to an overall reduction in execution time. For both versions of the BDNMiner algorithm, the execution time decreases as $PIncT$ increases due to the decrease in the number of non-compliant windows and consequently the number of enumerated patterns. We can also see that between 0 % and 50 %, BDNMiner does not show the same trend as MTNMiner since BDNMiner's execution time was less affected by the changes in the cross-K function values due to the simultaneous use of the minimum support pruning filter.

Figure 11l shows the execution times when varying $avgNO_xT$ from the EPA standard threshold of 0.267 up to 1.5 gm/kW-h. Similar to the effect of $PIncT$, as $avgNO_xT$ increases, fewer non-compliant windows are identified, leading to a decrease in the execution time for both BDNMiner and MTNMiner. However, as before, BDNMiner always performs significantly better.

8 Conclusion and future work

This work explored the problem of Non-compliant Window Co-occurrence (NWC) pattern discovery in relation to an important real-world application, eco-friendly transportation. The NWC discovery problem is challenging due to the large number of candidate patterns, large data volume and the lack of monotonicity in the temporal cross-K function used to measure the interestingness of a pattern. In this paper, we proposed a bi-directional pruning approach for mining NWC patterns (BDNMiner) which uses both cross-K function and minimum support pruning simultaneously. We also proposed a method for calculating tighter bounds for the cross-K function in BDNMiner as opposed to the bounds proposed in our preliminary work. For large datasets, the proposed BDNMiner algorithm was shown to be an order of magnitude faster. We also presented two case studies using engine measurement data that showed the effectiveness of the proposed algorithm in finding patterns of interest to engine scientists and which motivate future engine research.

In the future, we plan to relax some of our assumptions in the NWC problem by exploring patterns of variable lengths, particularly for datasets collected at higher frequencies, and considering different lag values for the different explanatory variables. In addition, we plan to explore more spatial aspects of the NWC discovery problem (e.g. the effect of left/right turns on non-compliant engine emissions). Moreover, we will investigate the discovery of statistically significant NWC patterns as well as a parallel formulation for BDNMiner to further enhance its scalability.

Acknowledgments This material is based upon work supported by the National Science Foundation under Grant No. 1029711, IIS-1320580, 0940818 and IIS-1218168, the USDOD under Grant No. HM1582-08-1-0017 and HM0210-13-1-0005, Ford University Research Program (URP), and the University of Minnesota under the OVPR U-Spatial and Minnesota Supercomputing Institute (MSI) (www.msi.umn.edu).

Appendix A: Proofs of preliminary results section

Lemma 1 *Given an NWC pattern C and a time lag δ, $Upper_{Loc}(|C \overset{\delta}{\bowtie} W_N|)$ is an upper bound of $|C \overset{\delta}{\bowtie} W_N|$*

Proof For every NWC pattern $\{S_i\}$ consisting of a single event-sequence where $\{S_i\} \subseteq$ C, $1 \leq i \leq Dim(C)$, we have $|\{S_i\}| \geq |C|$, where $|\{S_i\}|$ and $|C|$ are the cardinality of the patterns $\{S_i\}$ and C in the time series, respectively. Since $\{S_i\} \subseteq$ C, we also have $|\{S_i\} \overset{\delta}{\bowtie} W_N| \geq |C \overset{\delta}{\bowtie} W_N|$, $\forall\ 1 \leq i \leq Dim(C)$. Then, $Upper_{Loc}(|C \overset{\delta}{\bowtie} W_N|) = \min_{\{S_i\} \in C, 1 \leq i \leq Dim(C)} (|\{S_i\} \overset{\delta}{\bowtie} W_N|) \geq |C \overset{\delta}{\bowtie} W_N|$. □

Lemma 2 *Given an NWC pattern C and a time lag δ, $Lower(|C|)$ is a lower bound of $|C|$.*

Proof Any superset pattern of C has a cardinality smaller than or equal to C. Therefore, Lower($|C|$) = |superset(C)| $\leq |C|$. □

Theorem 1 *Given an NWC pattern C and a time lag δ, $UB_{local}(\hat{K}_{C,W_N}(\delta))$ is an upper bound of $\hat{K}_{C,W_N}(\delta)$.*

Proof Using Lemmas 1 and 2, we have $\hat{K}_{C,W_N}(\delta) = \frac{T}{|W_N|} \times \frac{|C \overset{\delta}{\rhd\lhd} W_N|}{|C|} \leq \frac{T}{|W_N|} \times \frac{Upper_{Loc}(|C \overset{\delta}{\rhd\lhd} W_N|)}{Lower|C|} = UB_{local}(\hat{K}_{C,W_N}(\delta))$ □

Theorem 2 *Given an NWC pattern C and a time lag δ, $UB_{lattice}(\hat{K}_{C,W_N}(\delta))$ is an upper bound of $\hat{K}_{C,W_N}(\delta)$.*

Proof Since Definition 8 differs from Definition 7 only in the min term being replaced by a max term, the proof of Theorem 2 is straightforward from Theorem 1. □

Lemma 3 *Given an NWC pattern C and a time lag δ, $UB_{lattice}(\hat{K}_{C,W_N}(\delta))$ is monotonically decreasing with decreasing Dim(C) if $Lower(|C|)$ is kept monotonically increasing. In other words, given two NWC patterns C and C' where C' ⊂ C, then if $Lower(|C'|) \geq Lower(|C|)$, then $UB_{lattice}(\hat{K}_{C',W_N}(\delta)) \leq UB_{lattice}(\hat{K}_{C,W_N}(\delta))$.*

Proof Let C′ ⊂ C where C and C′ are two NWC patterns. Then ∀ $S_i(v_i) \in C'$, where $1 \leq i \leq \text{Dim}(C')$, we have $S_i(v_i) \in C$. Therefore, $Upper_{Lat}(|C' \overset{\delta}{\bowtie} W_N|) = \max_{\{S_i\} \in C', 1 \leq i \leq Dim(C')}(|S_i \overset{\delta}{\bowtie} W_N|) \leq \max_{\{S_i\} \in C, 1 \leq i \leq Dim(C)}(|S_i \overset{\delta}{\bowtie} W_N|) = Upper_{Lat}(|C \overset{\delta}{\bowtie} W_N|)$ (A). Also, since $Lower(|C|)$ is kept monotonically increasing as Dim(C) decreases, then $Lower(|C'|) \geq Lower(|C|)$ (B). From (A) and (B), we have $UB_{lattice}(\hat{K}_{C,W_N}(\delta)) = \frac{T}{|W_N|} \times \frac{Upper_{Lat}(|C \overset{\delta}{\rhd\lhd} W_N|)}{Lower(|C|)} \geq \frac{T}{|W_N|} \times \frac{Upper_{Lat}(|C' \overset{\delta}{\rhd\lhd} W_N|)}{Lower(|C'|)} = UB_{lattice}(\hat{K}_{C',W_N}(\delta))$ □

References

1. Aggarwal CC, Bhuiyan MA, Hasan MA (2014) Frequent pattern mining algorithms: A survey. In: Frequent pattern mining, Springer
2. Agrawal R, Srikant R et al. (1994) Fast algorithms for mining association rules. In: Proc. 20th int. conf. very large data bases, VLDB, vol 1215, pp 487–499
3. Ali RY, Gunturi VM, Kotz AJ, Shekhar S, Northrop WF (2015) Discovering non-compliant window co-occurrence patterns: A summary of results. In: Advances in Spatial and Temporal Databases. Springer, pp 391–410
4. Assanis DN, Filipi ZS, Fiveland SB, Syrimis M (2003) A predictive ignition delay correlation under steady-state and transient operation of a direct injection diesel engine. J Eng Gas Turbines Power 125(2):450–457
5. Cohen E, et al. (2001) Finding interesting associations without support pruning. IEEE Trans Knowl Data Eng 13(1):64–78
6. Das G, et al. (1998) Rule discovery from time series. In: Proceedings of the ACM International Conference on Knowledge and Data Discovery, pp 16–22
7. Daw CS, Finney CEA, Tracy ER (2003) A review of symbolic analysis of experimental data. Rev Sci Instrum 74(2):915–930
8. DieselNet (2015) Heavy-Duty Onroad Engines. https://www.dieselnet.com/standards/us/hd.php
9. Diggle PJ, Chetwynd AG, Häggkvist R, Morris SE (1995) Second-order analysis of space-time clustering. Stat Methods Med Res 4(2):124–136
10. Dixon PM (2002) Ripley's k function. Encyclopedia of environmetrics
11. FH Administration (2014) Annual vehicle distance traveled in miles and related data - 2011. https://www.fhwa.dot.gov/policyinformation/statistics/2011/pdf/vm1.pdf
12. Fang K, Li Z, Shenton A, Fuente D, Gao B (2015) Black box dynamic modeling of a gasoline engine for constrained model-based fuel economy optimization. Tech. rep., SAE Technical Paper
13. Gabriel E, Diggle PJ (2009) Second-order analysis of inhomogeneous spatio-temporal point process data. Statistica Neerlandica 63(1):43–51

14. Harms SK, Deogun JS (2004) Sequential association rule mining with time lags. J Intell Inf Syst 22(1):7–22
15. Huang Y, et al. (2003) Mining confident co-location rules without a support threshold. In: Proceedings of the 2003 ACM symposium on Applied computing, pp 497–501
16. Kotsiantis S, Kanellopoulos D (2006) Discretization techniques: a recent survey. GESTS Intl Trans on Comput Sci Eng 32(1):47–58
17. Lin J, Keogh E, Wei L, Lonardi S (2007) Experiencing sax: a novel symbolic representation of time series. Data Min Knowl Disc 15(2):107–144
18. McIntosh T, Chawla S (2007) High confidence rule mining for microarray analysis. IEEE/ACM Trans Comput Biol Bioinform 4(4):611–623
19. McKibben B (2014) Climate change impacts in the united states: the third national climate assessment
20. Misra C et al. (2013) In-use nox emissions from model year 2010 and 2011 heavy-duty diesel engines equipped with aftertreatment devices. Environ Sci Tech 47(14):7892–7898
21. Office of Transportation & Air Quality (2014) Mpg: Label values vs. corporate average fuel economy (cafe) values label mpg
22. Sacchi L, Larizza C, Combi C, Bellazzi R (2007) Data mining with temporal abstractions: learning rules from time series. Data Min Knowl Disc 15(2):217–247
23. Schiermeier Q (2015) The science behind the volkswagen emissions scandal. Nature
24. Schluter T, Conrad S (2011) About the analysis of time series with temporal association rule mining. In: IEEE Symposium on Computational Intelligence and Data Mining, pp 325–332
25. Shen W, Wang J, Han J (2014) Sequential pattern mining. In: Frequent pattern mining. Springer
26. Turns SR (2012) An Introduction to Combustion: Concepts and Applications, vol 287, 3rd. McGraw-hill, New York
27. US Environmental Protection Agency (2014a) Ground level ozone health effects
28. US Environmental Protection Agency (2014b) Inventory of U.S. Greenhouse Gas Emissions and Sinks: 1990 - 2012. https://www3.epa.gov/climatechange/ghgemissions/usinventoryreport.html
29. US Government Publishing Office (2015) 40 cfr ch. u section 1036.108. http://goo.gl/fg5NyV
30. Vijayaraghavan K et al. (2012) Effects of light duty gasoline vehicle emission standards in the United States on ozone and particulate matter. Atmos Environ 60:109–120
31. Wang J, He QP (2010) Multivariate statistical process monitoring based on statistics pattern analysis. Ind Eng Chem Res 49(17):7858–7869
32. Wikipedia (2015) Sudden unintended acceleration. https://goo.gl/OvMi6w
33. Zakaria W, Kotb Y, Ghaleb F (2014) Mcr-miner: Maximal confident association rules miner algorithm for up/down-expressed genes. Appl Math 8(2):799–809

Reem Y. Ali is a Ph.D. candidate at the Department of Computer Science and Engineering, University of Minnesota - Twin Cities. She received her Bachelor's and Master's degrees in Computer Science from Alexandria University in Egypt in 2008 and 2012, respectively. Her research interests include spatial/spatio-temporal data mining and spatial/spatio-temporal databases. For her research on connected cars, she has been awarded the ACM SIGSPATIAL 2015 Second Best Vision Paper award and her work has been selected as one of SSTD 2015 Best Papers.

Venkata M.V. Gunturi is an assistant professor at the Indraprastha Institute of Information Technology, Delhi, India. He completed his PhD in Computer Science from the University of Minnesota Twin Cities, USA in 2015. Prior to PhD, he completed an MTech degree in Computer Science from IIT Kanpur, India in 2010. His research interests include Spatial and Spatio-temporal databases, Graph Algorithms and Geographic Information Science.

Andrew J. Kotz is a Ph.D. candidate in the Department of Mechanical Engineering at the University of Minnesota working in the TE Murphy Engine Research Laboratory. He received his B.E. in Mechanical Engineering in 2012 from the Steven's Institute of Technology where he received the Fernando Sisto Award for excellence in aerospace and turbomachinery and was the captain of the Formula SAE team. He received his M.S. in mechanical engineering from the University of Minnesota in 2015 and is expected to graduate with his Ph.D. in mechanical engineering in 2016. Throughout his graduate career Andrew has received the Mathew J Hubber award for excellence in transportation research and education and is MnDRIVE Scholar for robotics outreach.

Emre Eftelioglu is a Ph.D. candidate at the Department of Computer Science and Engineering, University of Minnesota - Twin Cities. He received his Bachelor's degree in Turkey (2004) in Systems Engineering and his Master's degree at the University of Minnesota (2014) in Computer Science. His research interests include spatial data mining and spatial databases.

Shashi Shekhar is a McKnight Distinguished University Professor at the University of Minnesota. He co-edited an Encyclopedia of GIS and co-authored a Spatial Databases textbook. He received the IEEE-CS Technical Achievement Award, the UCGIS Education Award and was elected a Fellow of the IEEE and the AAAS. Shashi is a co-Editor-in-Chief of Geo-Informatica journal (Springer) and has served on the Computing Community Consortium Council and National Academies committees.

William F. Northrop is an assistant professor in mechanical engineering at the University of Minnesota and a principal investigator at the TE Murphy Engine Research Laboratory. He received M.S. and Ph.D. degrees in mechanical engineering from University of Michigan in 2003 and 2009, respectively and a B.S in mechanical engineering from Carnegie Mellon University in 1997. For his research in combustion and emissions related to internal combustion engines, he was the recipient of the NSF CAREER Award in 2014, the University of Minnesota McKnight Land Grand Professorship in 2015 and Society of Automotive Engineers Ralph Teetor Award in 2016.