Efficient Search for Multi-Scale Time Delay Correlations in Big Time Series

Ho, Nguyen Thi Thao; Pedersen, Torben Bach; Ho, Long Van; Vu, Mai

Published in:
The 23rd International Conference on Extending Database Technology (EDBT), March 30-April 2, 2020

Creative Commons License
CC BY-NC-ND 4.0

Publication date:
2020

Document Version
Accepted author manuscript, peer reviewed version

Link to publication from Aalborg University

Citation for published version (APA):
Efficient Search for Multi-Scale Time Delay Correlations in Big Time Series

Nguyen Ho‡, Torben Bach Pedersen†, Van Long Ho†, Mai Vu‡
†Aalborg University, ‡Tufts University

ABSTRACT

Very large time series are increasingly available from an ever wider range of IoT-enabled sensors deployed in different environments. Significant insights and values can be obtained from these time series through performing cross-domain analyses, one of which is analyzing time delay temporal correlations across different datasets. Most existing works in this area are either limited in the type of detected relations, e.g., linear relations alone, only working with a fixed temporal scale, or not considering time delay between time series. This paper presents our Time delay CORelation Search (TYCOS) approach which provides a powerful and robust solution with the following features: (1) TYCOS is based on the concept of mutual information (MI) from information theory, giving it a strong theoretical foundation to detect all types of relations including non-linear ones. (2) TYCOS is able to discover time delay correlations at multiple temporal scales, (3) TYCOS works in an efficient, bottom-up fashion, pruning non-interesting time intervals from the search by employing a novel MI-based theory, and (4) TYCOS is designed to efficiently minimize computational redundancy. A comprehensive experimental evaluation using synthetic and real-world datasets from the energy and smart city domains shows that TYCOS is able to find significant time delay correlations across different time intervals among big time series. The performance evaluation shows that TYCOS can scale to large datasets, and achieve an average speedup of 2 to 3 orders of magnitude compared to the baselines by using the proposed optimizations.

1 INTRODUCTION

Rapid advancements in IoT technology have enabled the collection of enormous amounts of time series data at unprecedented scale and speed. For example, a modern wind turbine has hundreds of sensors sampled at a high frequency, a smart building contains thousands of sensors sensing the surrounding environment, and an autonomous vehicle carries numerous vision sensors. All of them are collecting terabytes of data everyday. Analyzing these massive, heterogeneous and rich datasets can help uncover hidden patterns and extract new insights to support evidence-based decision making.

While time series analysis has been studied extensively in the past, its importance and value only continue to grow. One of the first steps to harness the enormous potential from modern big time series is to discover correlations among heterogeneous and cross-domain datasets. Consider for example the NYC Open Data [2] with more than 1,500 published datasets containing quantitative data from different domains, including weather and transportation, energy and environment etc. Cross-domain analysis among these datasets can reveal new insights about the city and its citizens, and thus aid policy makers in decision making.

For instance, finding correlations between weather and transportation data can lead to the identification of individual weather events, such as the occurrence of a storm or a hurricane, which then helps explain an abnormal increase in the number of accidents. Data correlation is also useful in behavioral prediction and future planning. For example, illustrating that weather data (e.g., wind speed) is well-correlated with energy production can provide accurate prediction of the city’s energy capacity at a specific time, thus allowing better resource planning. In the financial domain, data correlation can help forecast the price movement of related stocks, or predict the purchasing behavior of consumers, and thus assist investors in making real-time investment decisions. Not only is it useful in reasoning and predicting, data correlation can also be considered as one of the three building blocks to establish a causal relation [3], and thus can serve as a basis for constructing inference and learning models.

Despite its potential use, finding correlations in big time series is challenging. Not only does the very large volume of data raise significant challenges in terms of performance and scalability, their complex and noisy nature also presents difficulties in finding different types of correlation relations, or in the ability to deal with adaptive temporal scales. For example, stock prices or weather data exhibit non-linear relations, which cannot be captured by traditional correlation metrics such as Pearson Correlation Coefficient [23]. Besides, there is often a misconception that finding correlations and finding similarities in time series are the same task, where in fact, they are two different problems. Finding correlations is to look for statistical relationships in the data, whereas finding similarities means to find the optimal matching and/or alignment between time series sequences. Unlike the correlation-based approach, similarity metrics (which have positive values only) cannot distinguish between non-correlated and negatively correlated time series, which will both have values close to 0. For example, consider a pair of time series \((X, Y)\) generated by a sine function \(y = \sin(x)\). Here, \(x \in (-\infty, \infty)\) represents a linearly increasing time series, while \(Y\) follows a sine function of \(X\). In this example, \(X\) and \(Y\) do not exhibit any similarities among their values, but they do have an underlying relation. Such non-linear relations are common in areas such as signal processing, but cannot be detected using similarity measures. Thus, methods such as those used in Dynamic Time Warping [28] or MatrixProfile [31] have significant limitations in analyzing modern time series.

To make the problem even more challenging, cross-domain correlations might appear at different temporal scales. For example, correlations involving weather data might span over multiple temporal durations ranging from hours (e.g., during rain showers), to days, or even weeks (e.g., during a storm) depending on the weather events. Likewise, interactions between events might not always occur simultaneously. In practice, it is common to see events of one phenomenon influence other phenomena only after some delay of time. For instance, an increase of incidents caused by heavy rain can only be observed minutes or hours after the rain starts; or the impact of one rising stock on other
Finding correlations among datasets is a fundamental step in data exploration. In the past, correlation analyses relied on traditional statistical metrics such as covariance or correlation coefficients to measure correlations [13, 15, 18, 19, 32]. However, these metrics are usually best for linear and/or monotonic dependencies. Recent studies have attempted to approach the problem from a high level. Sarma et al. [10] use the concept of relatedness, Pochampally et al. [24] use joint precision and joint recall, Alawini et al. [4] rely on the history and schema of datasets, Roy et al. [26] use the concept of intervention, to identify relations between datasets or data tables. Middelfart et al. [21] propose a bitmap-based approach to measure change relationships in a data cube. Chirigati et al. [7] propose a topology-based framework to identify spatio-temporal relationships in heterogeneous data corpuses. These studies, however, only focus on overall correlations. None of them consider correlations in time windows.

Surprisingly, very little effort has been made to design efficient solutions for time delay window-based correlations. Among them, Rakthanmanon et al. [25] design a Dynamic Time Warping-based technique (MASS) to quickly find the most similar subsequences in time series. Although considered to be the state of the art for subsequences matching, the technique does not have a mechanism to automatically search for correlated windows, but rather relies on a provided query. To improve MASS, Yeh et al. [31] designed the MatrixProfile framework to perform similarity joins between time series. However, as will be shown in Section 8.3, MASS and MatrixProfile cannot detect complex relations such as non-linear and non-functional ones. Other works, e.g., [8, 29] propose sliding window-based procedures to detect hidden correlations. However, they only focus on fixed size windows, not considering time delay, or using correlation coefficients as correlation measures, and thus, cannot find multi-scale time delay correlations and are limited in the types of relations they can detect. Our work in this paper overcomes those limitations. Since TYCOS uses MI as a correlation metric, it can discover all types of relationships. Furthermore, TYCOS works in a bottom-up fashion, and can thus automatically discover time delay correlations at multiple temporal scales.

Prior to this work, we investigated the use of MI in correlation discovery, and proposed AMIC [16, 17], a top-down approach to search for multi-scale correlations in big data. However, AMIC does not consider time delay correlations. Recently, we examine the power of LAHC in correlation search in a short paper [14]. The present paper significantly extends [14] by considering time delay correlations, and proposes a novel noise theory and MI computation technique to achieve better performance.

3 BACKGROUND

3.1 Mutual Information

MI is a statistical measure to quantify the shared information between two probability distributions. Given two discrete random variables $X$, $Y$ with the corresponding probability mass functions (p.m.fs) $p(x, y)$, and the joint distribution $p(x, y)$, the MI between $X$ and $Y$ is defined as

$$I(X; Y) = \sum_{x \in X} \sum_{y \in Y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}$$

Intuitively, $I(X; Y)$ represents the reduction of uncertainty of one variable (e.g., $X$) given the knowledge of another variable (e.g., $Y$) [9]. The larger $I(X; Y)$, the more information is shared between $X$ and $Y$, and thus, the less uncertainty about one variable when knowing the other. The property that MI is equal to zero if and only if the considered variables are statistically independent, otherwise always positive if there exists any kind of dependency (e.g., linear and non-linear) [11], makes MI a versatile measure to capture correlations in noisy datasets which often exhibit a high degree of bias and abnormality, causing their relationships to often be arbitrary and non-linear.

Estimating mutual information: Eq. (1) is the theoretical definition of MI but is usually not used for computing MI, as it requires having the distributions of the underlying data which are often unknown in practice. To estimate MI from collected samples, we choose an estimation method proposed by Kraskov et al. [20] (hereafter called the KSG method) for several reasons: (1) The KSG method outperforms other estimators (e.g., histogram, kernel density estimation) in terms of computational efficiency and
where each time step $t$.

The main idea of the KSG estimator is, instead of directly computing the joint and marginal probability distributions of the considered variables, it approximates the distributions by computing the densities of data points in nearby neighborhoods [20]. Specifically, KSG computes the probability distribution for the distance between each data point and its $k$th nearest neighbor. For each data point, it searches for $k$ nearest neighbor clusters ($k$ is a pre-defined parameter) and computes the distance $d_k$ to the $k$th neighbor. Then, the population density is estimated by counting the number of data points that fall inside $d_k$. This leads to the computation of MI between two variables $X$ and $Y$ as [20]:

$$I(X; Y) = \psi(k) - 1/k - \psi(n_x) - \psi(n_y) + \psi(n)$$

where $\psi$ is the digamma function, $k$ is the number of nearest neighbors, $(n_x, n_y)$ are the number of marginal data points in each dimension falling within the distance $d_k$, $n$ is the total number of data points and $(\cdot)$ is the average function. The digamma function $\psi$ is a monotonically increasing function. Thus, the larger $n_x$ and $n_y$ (i.e., more data points fall into the distance $d_k$), the lower $I(X; Y)$, and vice versa.

### 3.2 Late Acceptance Hill Climbing

Our correlation search algorithm is built based on LAHC [6] which we briefly introduce next. LAHC is an optimization technique attempting to find local optimal solutions for a given problem through iterative improvement. Given a target function $f$ and a current solution $S$ of $f$, LAHC tries to improve $S$ by exploring potential candidates in the nearby neighborhood. If a better solution for $f$ is found (according to some criteria), the current solution $S$ is replaced by this new solution $S_{new}$, and the process is repeated until no further improvement can be made. LAHC is an extension of the classic Hill Climbing (HC) [27], but it differs from HC in its acceptance rule: a solution $S_{new}$ is accepted if $S_{new}$ is better than either the current solution $S$ or a solution $S_{old}$ found in the history. To do that, LAHC uses a fixed length array $L_k$ to maintain a history of the most recent accepted solutions, and use $L_k$ to justify the goodness of a candidate solution.

### 4 PROBLEM FORMULATION

**Definition 4.1** (Time series) A time series $X_T = \{x_1, x_2, \ldots, x_n\}$ is a sequence of data values that measures the same phenomenon during an observation period $T$, and is sorted in time order.

Note that the time period $T = [t_1, t_n]$ contains $n$ time steps where each time step $t_i$ has a recorded value $x_i$ in $X_T$, and $t_1$ and $t_n$ denote the first and the last time step of $T$. We say $X_T$ has length $n$ if $X_T$ contains $n$ data samples.

**Definition 4.2** (Time window) A time window $w = [t_s, t_e]$ is a temporal sub-interval of $T$ that records the events of $X_T$ from time step $t_s$ to time step $t_e$, and forms a (sub) time series $X_w = \{x_{t_s}, \ldots, x_{t_e}\} \subset X_T$.

We say $w$ has size $m$, denoted as $|w| = m$, if $w$ contains $m$ time steps, and is equivalent to $X_w$ containing $m$ data samples.

**Definition 4.3** (Pair of time series) A pair of two time series $(X_T, Y_T) = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ contains data collected from $X_T$ and $Y_T$ that measure two separate phenomena during the same observation period $T$. A tuple $(x_i, y_i) \in (X_T, Y_T)$ records the data values on $X_T$ and $Y_T$ at the same time step $t_i$.

**Definition 4.4** (Pair of time windows) Let $w_X = [t_s, t_e]_X$, $w_Y = [t_y, t_\tau]_Y$ be time windows of $X_T$ and $Y_T$, respectively. Assume $w_X$ and $w_Y$ have the same length, i.e., $|w_X| = |w_Y|$. The pair of time windows $(w_X, w_Y)$ is $\{(t_s, t_{\tau}^1), \ldots, (t_s, t_{\tau}^k)\}$ records the events of $X_T$ from $(t_s, t_{\tau}^1)$, and of $Y_T$ from $(t_y, t_{\tau}^k)$, and forms a pair of (sub) time series $(X_w, Y_w) = \{(x_{t_{\tau}^1}, \ldots, x_{t_{\tau}^k}), (y_{t_y}, \ldots, y_{t_{\tau}^k})\} \subseteq (X_T, Y_T)$.

**Definition 4.5** (Time delay window of a time series pair) Let $(w_X, w_Y) = \{(t_s, t_e^1), \ldots, (t_s, t_e^k)\}$ be a pair of time windows like in Definition 4.4, and $\tau$ be an integer. The pair $(w_X, w_Y)$ is called a time delay window of $(X_T, Y_T)$ with the delay $\tau$ if $t_{\tau}^i = t_{s} + \tau$, and is denoted as $w_{X,Y,\tau} = \{(t_s, t_{\tau}^i, \tau)\}$, where $t_\tau^1 = t_s$, and $t_{\tau}^i = t_{\tau}^{i-1} + \tau$, are the start time and the end time of $w_{X,Y,\tau}$ on $X_T$, and $\tau$ is the time delay of $w_{X,Y,\tau}$.

The window $w_{X,Y,\tau} = \{(t_s, t_{\tau}^i), \tau\}$ in Definition 4.5 defines a one-to-one mapping $f$: $w_X \mapsto w_Y$ that maps each event in $w_X$ to the corresponding event in $w_Y$. The mapping is time correspondence, i.e., the event at the $i$th time step of $X_T$ in $w_X$ is mapped to the event at the $(i + \tau)$th time step of $Y_T$ in $w_Y$. Each window $w_{X,Y,\tau}$ is characterized by three parameters: the start time $t_s$, the end time $t_{\tau}^i$, and the time delay $\tau$. The size of $w_{X,Y,\tau}$ equals to the size of $w_X$ and $w_Y$, i.e., $|w_{X,Y,\tau}| = |w_X| = |w_Y|$.

A time delay window represents a shift (also called "delay" or "lag") in time between two time series $X_T$ and $Y_T$, and the value of $\tau$ indicates the shifted time units. Since $\tau$ can be equal to 0, or positive, or negative, the window $w_{X,Y,\tau}$ is called time delay windows for all time shifting scenarios. Semantically, if $\tau = 0$, then $w_{X,Y,\tau}$ does not have a time delay (or events of $X_T$ in $w_X$ and events of $Y_T$ in $w_Y$ occur at the same time). Whereas if $\tau > 0$, then $w_{X,Y,\tau}$ is delayed $\tau$ time units from $w_X$ (or events in $w_Y$ occur $\tau$ time units after events in $w_X$). Similarly, if $\tau < 0$, $w_{X,Y,\tau}$ is delayed $\tau$ time units from $w_Y$.

**Example 1** Consider a pair of time series (Rain Precipitation (RP), Injured Pedestrian (IP)), and a time window $w_{RP,IP,30} = \{9:00\ am, 10:00\ am, 30\ mins\}$. The window $w_{RP,IP,30}$ contains events of RP during [9:00 am, 10:00 am], and maps them to events of IP occurring 30 minutes later, i.e., during the interval [9:30 am, 10:30 am].

Fig. 1 illustrates 3 different scenarios of time window on $(X_T, Y_T)$. Here, $w_1 = \{(t_1, t_2), \tau_1 = 0\}$ has no time delay, thus starts and ends at the same time on $X_T$ and $Y_T$. Instead, the window $w_2 = \{(t_3, t_5), \tau_2 > 0\}$ has a time delay $\tau_2 > 0$, thus $Y_T$ is shifted from $X_T$. The window $w_3 = \{(t_5, t_7), \tau_3 < 0\}$ has $\tau_3 < 0$, thus $X_T$ is shifted from $Y_T$, similarly for $w_3$.

**Definition 4.6** (Mutual information of a window) Let $(X_T, Y_T)$ be a pair of time series, and $w_{X,Y,\tau}$ be a time delay window of $(X_T, Y_T)$. The MI between $X_T$ and $Y_T$ within $w_{X,Y,\tau}$ is estimated using the KSG estimator as:

$$I(w_{X,Y,\tau}) = I(X_w; Y_w) = \psi(k) - 1/k - 1/m \sum_{y \in Y_w} |\psi(nx_y) + \psi(ny_y)| + \psi(m)$$

where $m$ is the size of $w_{X,Y,\tau}$, and $n_x$, and $n_y$, are the number of data points falling within the $k$th-nearest distances in each dimension $d_x$ and $d_y$ of point $(x_i, y_i) \in (X_w, Y_w)$.
{p1=(x1, y1), ..., p2=(xN, yN)}, with their positions projected into a two-dimensional grid as in Fig. 2. Without loss of generality, we assume τ = 0 (events in Xw and Yw occur at the same time step), the nearest neighbor parameter k = 2, and the distance metric between neighbors is the maximum norm \( ||.|| \). Under this setting, the 2 nearest neighbors of \( p_1 \) are \( p_2 \) and \( p_3 \) (in green), and the 3 nearest distances from \( p_1 \) to its nearest neighbors in each dimension are \( dx \) and \( dy \). The nearest distances allow the KSG estimator to form the marginal regions (in gray shade), from which the marginal counts are computed. In this case for point \( p_1 \), the marginal counts are \( n_{x1} = 3 \) (including \( p_2, p_3, p_4 \)), and \( n_{y1} = 3 \) (including \( p_2, p_3, p_4 \)). Similar steps are applied to other data points from \( p_2 \) to \( p_7 \). Finally, the marginal counts \( n_{x1}, n_{y1} \) are inserted into Eq. 3 to compute the MI of \( w_{x1, y1} \).

**Problem Statement:** Time delay Correlation Search (TYCOS). Let \((X_T, Y_T)\) be a pair of time series measured during the time interval \( T \), and \( w_{X,Y} \) be a time delay window of \((X_T, Y_T)\). Then TYCOS aims to find a set \( S \) of \( w_{X,Y} \) such that \( s_{\min} \leq |w_{X,Y}| \leq s_{\max} \) and \( \tau \leq \tau_{\max} \). Furthermore, each window \( w_i \) has the possibility to shift \( 2 \times t_{\max} \) times (corresponding to negative and positive values of \( \tau \)), creating \( 2 \times t_{\max} \) possible time delay windows. Finally, there are \( n \times (s_{\max} - s_{\min} + 1) \) possible start indices \( t_{\max} \). Thus, the total number of feasible windows of TYCOS is:

\[
(n - s_{\min} + 1) \times (s_{\max} - s_{\min} + 1) \times 2 \times t_{\max} \sim n^3
\]

If \( s_{\max} \to n \) and \( t_{\max} \to n \), and \( s_{\min} \ll n \).

**Lemma 2.** Let \( n \) be the length of \((X_T, Y_T)\), and \( m \) be the average size of a window, then the worst-case time complexity of a brute force search for TYCOS on \((X_T, Y_T)\) is \( O(n^3m^2) \).

**Proof.** The complexity of TYCOS depends on the number of windows it needs to evaluate, and the time required to compute the MI of each window. The number of windows to be evaluated for TYCOS is \( O(n^3) \), according to Lemma 1.

5. **TYCOS: Time Delay Correlation Search**

5.1 **Search Space and Time Complexity**

The search space of TYCOS is represented by the number of feasible windows (feasible windows are those that respect the size and time delay constraints), illustrated in Fig. 3. Here, the \( x \)-axis represents the start time \( t_s \), the \( y \)-axis represents the end time \( t_e \), and the \( z \)-axis represents the time delay \( \tau \) of a window. Each point in this three-dimensional grid represents a window \( w_i \) identified by its start time index \( t_{s, i} \), end time index \( t_{e, i} \), time delay \( \tau_{i} \), and its MI \( I_{w, i} \). Since the start time index \( t_{s, i} \) always has to be smaller than the end time index \( t_{e, i} \), the feasible windows will reside only in half of the grid (Fig. 3).

**Lemma 1.** Let \((X_T, Y_T)\) be a pair of two time series of length \( n \), and \( s_{\min}, s_{\max} \) be the minimum and maximum sizes of a window, \( t_{\max} \) be the maximum time delay between \( X_T \) and \( Y_T \). Then the size of TYCOS search space is \( O(n^3) \).

**Proof.** To find all feasible windows, initially, a Brute Force search can start with a window \( w_0 = ([t_{s, 0}, t_{e, 0}], 0) \) at the minimum size \( s_{\min} \) and the initial time delay \( \tau = 0 \). For each start index \( t_{s, i} \), it extends the end index \( t_{e, i} \), creating a new and larger window \( w_i \) until it reaches the maximum size \( s_{\max} \). With one start index \( t_{s, i} \), the number of windows created by extending the end index is \( (s_{\max} - s_{\min} + 1) \). Furthermore, each window \( w_i \) has the possibility to shift \( 2 \times t_{\max} \) times (corresponding to negative and positive values of \( \tau \)), creating \( 2 \times t_{\max} \) possible time delay windows. Finally, there are \( n \times (s_{\max} - s_{\min} + 1) \) possible start indices \( t_{s, i} \). Thus, the total number of feasible windows of TYCOS is:

\[
(n - s_{\min} + 1) \times (s_{\max} - s_{\min} + 1) \times 2 \times t_{\max} \sim n^3
\]
is $O(n^2 m^2)$. However, if a more efficient data structure is used, such as k-d tree [5] or grid-based structure (for low dimensional data) [30], the expected-case $kNN$ complexity is $O(kd m \log m) \sim O(m \log m)$, and thus, the expected-case Brute Force complexity is $O(n^2 m \log m)$.

5.2 TYCOSLAHC: A LAHC Based Approach

The time complexity of a Brute Force approach for TYCOS is computationally prohibitive in practice. For example, a pair of time series with $n=9,000$ data points, $s_{\text{max}} = 400$, $s_{\text{min}} = 20$, and $td_{\text{max}} = 20$ will create 136,870,440 windows. Our Brute Force search implemented in C++ and run on a standard PC will take more than 12 hours to process all generated windows. In the next section, we propose a heuristic search method using LAHC to speed up the TYCOS process.

To improve the TYCOS process, we look at two angles for improvements: (1) reducing the search space, and (2) optimizing the MI computation. To reduce the search space, we adopt LAHC, and propose a novel MI-based theory to prune unpromising windows. To optimize the MI computation, we design efficient data structures so that we can reuse the computation across windows. The following sections discuss the intuition behind our approach and detail how LAHC can be applied to TYCOS. The MI-based theory and its applicability to TYCOS are introduced in Section 6. The efficient MI computation is described in Section 7.

5.2.1 The choice of LAHC. To explain the intuition behind the LAHC-based method, consider Fig. 4 that illustrates the MI value fluctuation across windows. Here, the $y$-axis represents the MI values of corresponding time windows on the $x$-axis. Given the correlation threshold $\sigma$ (red line), the three windows which correspond to the three locally maximal points (in red) indicate highly correlated areas, and can be found by identifying the three peak (red) points in the search space. Since LAHC guarantees to achieve local optimal solutions, it becomes an ideal foundation for solving the TYCOS problem.

5.2.2 Apply LAHC to TYCOS. Indeed, finding correlations in time series means to find windows that maximize the MI. Thus, we consider the problem of searching for time delay correlations using LAHC, namely TYCOSLAHC (or TYCOSL in short), as a maximization problem. Specifically, the target function of TYCOSL is a maximize function, and our goal is to find windows where their MIs are locally maximal values that satisfy $\sigma$.

a) Search space navigation. We first illustrate how LAHC navigates through the search space of TYCOS in Fig. 5, with the three axes being the start time ($x$-axis), the end time ($y$-axis) and the time delay ($z$-axis) of a window. Assume $w_i = ([t_{i_1}, t_{i_2}], t_{i_3})$ is the window where the search is currently at. Starting from $w_i$, if TYCOSL follows a rightwards trajectory on the $y$-axis, it moves the end time $t_{i_2}$ of $w_i$ forward in time, thus enlarging the window size. If it follows a leftwards trajectory on the $y$-axis, it moves the end time $t_{i_2}$ backward in time, thus reducing the window size. Similarly, moving along the $x$-axis, TYCOSL can reduce or increase the start time $t_{i_1}$, therefore, extending or narrowing the size of $w_i$ accordingly. On the $z$-axis, following the $t_{i_3}$ direction, TYCOSL increases the shifting time of $X_T$ w.r.t. $Y_T$. Following the $t_{i_3}$ direction, TYCOSL will shift $Y_T$ further from $X_T$. In both cases, it increases the time delay but keeps the same window size.

While exploring the search space in multiple directions, TYCOSL creates different windows by adjusting the indices of the current window. The generated windows are grouped into the same neighborhood if they share similar indices. The neighborhood concept is defined below.

**Definition 5.1** ($\delta$-neighbor) Let $w = ([t_{i_1}, t_{i_2}], \tau)$ be a window of $(X_T, Y_T)$, and assume $(X_T, Y_T)$ has length $n$. A window $w' = ([t_{i_1}', t_{i_2}'], \tau')$ is a $\delta$-neighbor of $w$ if $t_{i_1}' = t_{i_1} \pm \delta \lor t_{i_2}' = t_{i_2} \pm \delta \lor \tau' = \tau \pm \delta$, where $\delta$ is a pre-defined moving step such that $1 \leq \delta \leq n \land s_{\text{min}} \leq |w'| \leq s_{\text{max}} \land \tau \pm \delta \leq td_{\text{max}}$.

A $\delta$-neighbor window w' has at least one of its indices (i.e., $t_{i_1}'$, $t_{i_2}'$, or $\tau'$) differing $\delta$ step from the indices of w.

**Definition 5.2** ($\delta$-neighborhood) Let $w = ([t_{i_1}, t_{i_2}], \tau)$ be a window of $(X_T, Y_T)$. A $\delta$-neighborhood of $w$, denoted as $N_w$, is formed by all $\delta$-neighbors $w' = ([t_{i_1}', t_{i_2}'], \tau')$ of w.

The neighborhood concept is illustrated in Fig. 5. Consider the window $w_i$ (in blue). The nearest $\delta$-neighborhood of $w_i$, called the 1-neighborhood $N_{w_i}$, is the area formed by the 26 windows in blue color $w^j$ where $1 \leq i, j \leq 26$. Each window in this neighborhood differs from $w_i$ by one $\delta$ step, either by its start index, or its end index, or its time delay, or the combinations of them, or all. Going further, another neighborhood of $w_i$, called the 2-neighborhood $N_{2w_i}$, is the 50 windows in green color area. Each $\delta$-neighborhood forms an area where TYCOSL will iteratively look for potential candidates to improve $w_i$.

b) TYCOSL algorithm. We provide the outline of TYCOSL in Algorithm 1, and explain it in the following.

Consider a time series pair $(X_T, Y_T)$, and let $I_{w_i}$ be the target function to be maximized. To improve $I_{w_i}$, TYCOSL will start with an initial feasible solution, and explores its neighborhood to look for better solutions. Let $w = w_0$ where $|w_0| = s_{\text{min}} \land t_0 = 0$ be an initial solution (Alg. 1, line 2). The goodness of $w_0$ is evaluated by computing $I(w_0)$ (line 3). Starting from $w_0$, TYCOSL will first explore its nearest neighborhood $N_{w_0}$, and search for a better solution than $w_0$ in this area. To do that, it creates all $\delta_1$-neighbors of $w_0$ to form $N_{w_0}$. Then for each $w^j \in N_{w_0}$, it computes $I(w^j)$ and selects the best neighbor $bestnb$ which has the highest MI (lines 5 – 8). Next, it determines whether $bestnb$ is a better solution than the current one $w$ using the following policies:

- (Policy 1) If: $I_{bestnb} > I_{w} \lor I_{bestnb} > I_{w_0}$ where $w_0 \in L_k$, then: $bestnb$ is a better solution than $w$ and thus, $w$ is replaced by $bestnb$ (lines 10 – 12).

- (Policy 2) If: $I_{bestnb} \leq I_{w}$ and $I_{bestnb} \leq I_{w_0}$, then there is no better solution in the considered neighborhood, thus, no improvement can be made (lines 14 – 15).

In Policy 1, a better solution is found, the search moves to this new solution $w = bestnb$, and repeats the neighborhood exploration process on the new $w$. Note that since LAHC also uses a historical value $w_0$ to justify a potential candidate solution, the newly selected solution $bestnb$ might be better than $w_0$, but not necessarily better than the current solution $w$. This type of approximation creates some “randomness” in the search, which is helpful, for example, when the search needs to escape from plateau situations, i.e., when the search space is flat. In Policy 2, no better solution is found, then the stopping conditions are checked. If the stopping conditions are not yet satisfied, the search continues exploring larger neighborhoods. Otherwise, it stops and the value $I_{w}$ at the stopping point is the locally maximal value. Finally, $w$ is accepted and inserted into the result set $S$ if $I_{w} \geq \sigma$ (lines 19 – 20).

When the stopping conditions are satisfied and TYCOSL stops, the time series pair might not be scanned entirely. In that case, TYCOSL restarts again on the remaining part of the data, looking...
for new local optimal solutions, until the entire time series are searched (line 21).

Stopping conditions: Ideally, TYCOSL will stop immediately when no better solution can be found in the considered neighborhood. However, to avoid situations where the occurrence of a temporary setback stops the search too early, an idle period is used to measure the number of non-improvements observed. The search will stop when the pre-defined max idle period $T_{\text{maxidle}}$ is reached (line 4).

Initial solution: The initial window $w_0$ can be at the beginning, or at an arbitrary position in the time series. A good initial solution can help reach satisfying solutions faster, and vice versa. In Section 6, we rely on an MI-based theory to select a good initial solution, leading to a more promising exploration for the search.

The history list $L_h$: TYCOS maintains a history list $L_h$ of the most recently accepted solutions and uses it to justify the goodness of a potential candidate. In our implementation, TYCOS follows the random policy when selecting and updating an item in the history (line 9 and 16 – 18).

**Algorithm 1** TYCOSL: LAHC for TYCOS

```
Input: $(X_T, Y_T)$: pair of time series
Params: $\sigma, \epsilon, s_{\text{min}}, s_{\text{max}}, t_{\text{idle}}$
Output: $S$: a set of non-overlapping windows whose MI $\geq \sigma$

1. while $(X_T, Y_T)$ is not scanned entirely do
2. Initial solution $w := w_0$ with $|w_0| = s_{\text{min}}$ & $t_0 = 0$
3. Compute $H(w_0)$ → Evaluate the goodness of $w_0$
4. while $t_{\text{idle}} < T_{\text{maxidle}}$ do
5. $N := \text{Neighbors}(w)$ → Identify the neighbors of $w$
6. for $w' \in N$ do
7. Compute $H(w')$ → Evaluate the goodness of $w'$
8. $\text{bestnb} := \text{BestNeighbor}(N)$ → Select best neighbor in $N$
9. $w_0 := \text{random.get}(L_h)$ → Randomly select from $L_h$
10. if $I_{\text{best}} > I_{w'}$ or $I_{\text{bestnb}} > I_w$ then
11. $w := \text{bestnb}$ → Accept the candidate
12. $t_{\text{idle}} := 0$ → Reset the idle time
13. else
14. $w := w'$ → Reject the candidate
15. $t_{\text{idle}} := t_{\text{idle}} + 1$ → Increase the idle time
16. if $I_w > I_{\text{sub}}$ then
17. $w_0 := w'$ → Update the history list
18. $t_{\text{idle}} := t_{\text{idle}}$
19. if $I_w \geq \sigma$ then
20. Insert $w$ to $S$
21. TYCOSL$(X_T, Y_T)$ → Restart TYCOSL
22. return $S$
```

## 6 NOVEL NOISE THEORY TO IMPROVE TYCOS

### 6.1 Noise Identification

When TYCOSL performs the neighborhood exploration, conceptually, it is performing a depth-first search. Each neighbor window is considered as an expansion to a deeper level of the search tree, and the expansion only stops when the stopping conditions are met. During the expansion, some part of the data might be revisited multiple times, which can lead to redundant computation. To reduce potential redundancy, we explore several MI properties to establish principles that can help narrow the search space. Specifically, we seek the answer for the following research question: “When should a certain part of data be completely removed from the search?”

This research question concerns the removal of a data partition from the search without affecting its final outcomes. This is due to the fact that by repeatedly expanding the neighborhood, TYCOSL revisits a data partition multiple times, and in some cases, a particular data partition might be irrelevant to the search’s objectives, i.e., including this particular data into the search process does not lead to promising results. If that data partition can be identified, it should not be included in future explorations of the search. The following example demonstrates this situation.

Consider the window $w_1$ (blue point), and its neighborhood $N_1$ and $N_2$ in Fig. 5. In $N_1$ and $N_2$, neighbors that belong to the same exploration direction might contain overlapping data. For instance, $w_2 \in N_1$ is expanded from $w_1$ by extending its end index by a $\delta_1$ step, while $w_3 \in N_2$ is an extension of $w_1$ by enlarging $w_1$’s end index a $\delta_2$ step ($\delta_2 > \delta_1$). The process of extending one window to another window results in overlapping data that will be revisited multiple times in different exploration iterations.

On the other hand, consider Fig. 6 that plots the MI values of a time series pair with different start indices: the blue line starts at index 0, the red line starts at index 5, i.e., the data from 0 to 5 are not considered in the red line. From Fig. 6, it can be seen that by excluding the data range $[0-5]$ from the search, the MI values of subsequent windows increase and are larger than when including the considered range. This implies that the data range $[0-5]$ provides no information about the dependency between the time series pair, and thus can be considered as “noise” and eliminated from future exploration.

The above research question thus can be answered by establishing a “noise” identification principle. To do that, we rely on the following theorem to understand when a data partition can be considered as “noise” and should be eliminated.

**Definition 6.1 (Mixture distribution)** Let $X$ and $U$ be discrete random variables with the corresponding p.m.fs $p_X(x)$, $p_U(u)$. Let $Z$ be a new random variable which is drawn from the same distribution as $X$ with probability $\theta$ and from the same distribution as $U$ with probability $1-\theta$ for a given $\theta \in [0,1]$. Then $Z$ is said to have a mixture distribution between $p_X(x)$ and $p_U(u)$ with probability $\theta$ and is written as $Z = X \tilde{\otimes}_\theta U$.

**Theorem 6.1.** Let $X, Y, U, V$ be discrete random variables and $p_X(x), p_Y(y), p_U(u), p_V(v)$ be their corresponding p.m.fs.
We have the MI between $U$ and $W = Y \oplus V$ where $\oplus$ denotes the mixture of two variables. Assume that, except for $X$ and $Y$, all other variables are mutually independent, i.e., $U \perp V \land (X \perp U) \land (X \perp V) \land (Y \perp U) \land (Y \perp V)$. Then $I(X; Y) \geq I(Z; W)$.

**Proof.** $Z$ and $W$ are the two mixed variables: $Z = X \oplus \tilde{U}$ and $W = Y \oplus \tilde{V}$. Then, for a value of $x$ drawn according to $p_X(x)$ and a value of $u$ drawn according to $p_U(u)$, we can write the probabilities for $Z$ as follows:

$$p_Z(x) = P(Z = X)p_X(x) = \theta p_X(x) \quad (5)$$

$$p_Z(u) = P(Z = U)p_U(u) = (1 - \theta)p_U(u) \quad (6)$$

Similarly, we have:

$$p_W(y) = P(W = Y)p_Y(y) = \eta p_Y(y) \quad (7)$$

$$p_W(v) = P(W = V)p_V(v) = (1 - \eta)p_V(v) \quad (8)$$

Then, we can write the following joint probabilities:

$$p_{Z,W}(x, y) = \theta \eta p_{X,Y}(x, y) \quad (9)$$

$$p_{Z,W}(u, v) = (1 - \theta)\eta p_{U,V}(u, v) \quad (10)$$

$$p_{Z,W}(u, y) = (1 - \theta)(1 - \eta)p_{U,V}(u, y) \quad (11)$$

$$p_{Z,W}(u, v) = (1 - \theta)(1 - \eta)p_{U,V}(u, v) \quad (12)$$

We have the MI between $X$ and $Y$ as

$$I(X; Y) = \sum_{y} \sum_{x} p_{X,Y}(x, y) \log \frac{p_{X,Y}(x, y)}{p_X(x)p_Y(y)} \quad (13)$$

And the MI between $Z$ and $W$ as

$$I(Z; W) = \sum_{w} \sum_{z} p_{Z,W}(z, w) \log \frac{p_{Z,W}(z, w)}{p_Z(z)p_W(w)} \quad (14)$$

Since $Z$ can take the values in $\mathcal{K}_Z$ if $x$ is drawn from $X$, and in $\mathcal{K}_U$ if $z$ is drawn from $U$ (similarly for $W$), then from Eq. (14), it follows that:

$$I(Z; W) = \sum_{w \in \mathcal{K}_Z} \sum_{z \in \mathcal{K}_X} p_{Z,W}(x, y) \log \frac{p_{Z,W}(x, y)}{p_Z(x)p_W(y)} \quad (15)$$

This leads to:

$$I(U; Y) = 0 \land I(X; Y) = 0 \land I(U; V) = 0$$

Thus, Eq. (16) becomes

$$I(Z; W) = \theta \eta I(X; Y) \quad (17)$$

where $\theta \leq 1$ and $\eta \leq 1$. It leads to:

$$I(X; Y) \geq I(Z; W) \quad \Box$$

Theorem 6.1 says that, if $U$ and $V$ are independent from each other and from $X$ and $Y$, then adding them to $X$ and $Y$ will bring more uncertainty to $(X, Y)$, in other words, they reduce the shared information $I(X; Y)$.

**Definition 6.2 (Consecutive windows)** Let $w_{X,Y+\tau} = ([t_\tau, t_\tau], \tau)$ and $w_{X,Y+\tau}' = ([t_\tau', t_\tau'], \tau')$ be the two time delay windows of $(X_T, Y_T)$. Then $w_{X,Y+\tau}$ and $w_{X,Y+\tau}'$ are consecutive iff $t_\tau' = t_\tau + 1 \land \tau = \tau'$.

From Definition 6.2, $w_{X,Y+\tau}$ and $w_{X,Y+\tau}'$ are consecutive if they are next to each other and have the same shifting time, i.e., $w_{X,Y+\tau}'$ starts right after the end time of $w_{X,Y+\tau}$. Since $w_{X,Y+\tau}'$ follows $w_{X,Y+\tau}$, terminologically, we call $w_{X,Y+\tau}$ the followed window, and $w_{X,Y+\tau}'$, the following window. Examples of consecutive windows are $w_3$ and $w_4$ in Fig. 1.

**Definition 6.3 (Concatenation operation $\odot$ of consecutive windows)** Let $w_{X,Y+\tau} = ([t_\tau, t_\tau], \tau)$ and $w_{X,Y+\tau}' = ([t_\tau', t_\tau'], \tau')$ be two consecutive windows of $(X_T, Y_T)$. The concatenation between $w_{X,Y+\tau}$ and $w_{X,Y+\tau}'$ is defined as $w_{X,Y+\tau} \odot w_{X,Y+\tau}' = ([t_\tau, t_\tau'], \tau)$. The concatenation operation joins two consecutive windows $w_{X,Y+\tau}$ and $w_{X,Y+\tau}'$ into one bigger window $w_{X,Y+\tau}''$ which has its start time being the start time of the followed window, and its end time being the end time of the following window.

Based on the result of Theorem 6.1 and Definitions 6.2, 6.3, we define noise as follows.

**Definition 6.4 (Noise)** Let $w_{X,Y+\tau}$, $w_{X,Y+\tau}'$, be two consecutive windows of $(X_T, Y_T)$, $w_{X,Y+\tau}'' = w_{X,Y+\tau} \odot w_{X,Y+\tau}'$, be their concatenating window, and $\epsilon (0 \leq \epsilon < \sigma)$ be a real number representing the noise threshold. Assume that $I_{w_{X,Y+\tau}''} > 0$. Then $w_{X,Y+\tau}''$ is called noise w.r.t. $w_{X,Y+\tau}'$ iff $I_{w_{X,Y+\tau}'} - I_{w_{X,Y+\tau}''} > 0$.

The noise principle says that if the MI of the following window $w_{X,Y+\tau}''$ is less than the noise threshold, and the MI of the followed window $w_{X,Y+\tau}$ decreases after the concatenation, then the following window is noise w.r.t. the followed window.

### 6.2 Applying Noise Theory to Prune the Search Space

Based on the noise identification principle, we propose two improvements to be made in TYCOSLt. We name TYCOSLt with noise theory applied as TYCOSLN.
"valley-trapped" situations, we use the noise theory to find a good
starting point. The search is at a good starting point if the initial
solution \( w_0 \) has \( I_{w_0} \geq \varepsilon \) (the noise threshold). To find such a point, we
first divide the time series into non-overlapping windows of the
minimal size \( s_{\text{min}} \) with no time delay (\( \tau = 0 \)), and then
 hierarchically combine them to form larger, and hopefully better
windows. The combination stops when it finds a window \( w \) that
has \( I_w \geq \varepsilon \). Fig. 7 demonstrates this procedure.

In Step 1, the initial search starts with two minimal consecu-
tive and non-overlapping windows \( w_1, w_2 \), and evaluates
their goodness by computing \( I_{w_1}, I_{w_2} \). In Step 2, it combines the
two windows into a bigger one \( w_{12} \), and computes \( I_{w_{12}} \). Next, it
compares the goodness of the 3 windows, and select the one that
has the highest MI. Assuming that \( \{ I_{w_1}, I_{w_2} \} \leq I_{w_{12}} < \varepsilon \), then
\( w_{12} \) is the one selected among the three. Since \( I_{w_{12}} \) is still less
than \( \varepsilon \), it moves to Step 3.1, where a next minimal window \( w_3 \) is
evaluated both separately (by computing \( I_{w_3} \)), and together with
\( w_{12} \) (by computing \( I_{w_{12}w_3} \)).

Assume that \( I_{w_3} < \varepsilon \), and that by combining \( w_3 \) to \( w_{12} \), it re-
duces the MI \( I_{w_{12}w_3} \), i.e., \( I_{w_{12}w_3} < I_{w_{12}} < \varepsilon \). According to Theorem 6.1,
we can conclude that \( w_3 \) is noise w.r.t. \( w_{12} \). Thus, the combination
\( w_{12}w_3 \) does not lead to a promising result. The next window to
be considered is \( w_4 \). However, \( w_{12} \) cannot be combined with \( w_4 \)
without the presence of \( w_3 \), which we know is noise of \( w_{12} \). Thus,
the combination \( w_1w_2w_3 \) should not be formed, and \( w_{12} \) should also
be eliminated from future consideration (Step 3.3). Next, in Step 4,
\( w_3 \) is evaluated again in combination with \( w_4 \), and the procedure
is repeated until it can find a window that has MI \( \geq \varepsilon \). Once
the starting point is determined, TYCOSLN begins its neighborhood
exploration as described in Section 5.2.

6.2.2 Subsequent noise detection. The noise identification
principle is also beneficial during the neighborhood exploration.
We explain its applicability in Fig. 5. Assume \( w_1 \) is the current
window and \( w_1^1, w_1^2 \) are its neighbors when moving along the
\( y \)-axis. In the first exploration, the neighbor \( w_1^1 \) is considered.
Since \( w_1^1 \) is created by extending the end index of \( w_1 \) by a \( \delta_1 \)-step,
we have: \( w_1^1 = w_1 \circ w_6 \) where \( w_6 \) is the extension to be con-
catenated with \( w_1 \). Assume that by applying our noise theory to
\( w_1, w_6, \) and \( w_1^1 \), we conclude that \( w_6 \) is noise w.r.t. \( w_1 \). In this
case, it is not promising to further explore the neighborhoods of
\( w_1 \) along the \( y \)-axis in that direction. In the next exploration,
TYCOSLN will omit \( w_1^1 \), as well as the entire forward direction
along the \( y \)-axis.

**Ensuring the completeness of TYCOSLN:** When TYCOSLN stops
at a locally optimal solution, it has followed the best path and
explored to the deepest level of the current tree. This, however,
does not guarantee that is the only path. In fact, we want to find
the set of all windows that are above the correlation threshold.
Thus, to ensure the completeness of the search, TYCOSLN is
designed recursively so that once it stops at the locally optimal
solution, it goes back to the previously found starting point and
continues exploring other paths to find all feasible solutions.

Algorithm 2 reflects on how the noise theory is applied in
TYCOS. In line 2, the noise theory is applied to find a good
starting point. During the neighborhood exploration, the theory
is applied again to prune the search space (line 5).

**Algorithm 2 TYCOSLN: Apply noise theory to TYCOSL**

**Input:** \((X_T, Y_T)\) pair of time series

**Params:** \( \varepsilon, s_{\text{min}}, s_{\text{max}}, I_{\text{max}} \)

**Output:** \( S \) a set of non-overlapping windows whose MI \( \geq \sigma \)

1. **while** \((X_T, Y_T)\) is not scanned entirely **do**
2. **Initial solution** \( w := \text{InitialNoisePruning}((X_T, Y_T), \varepsilon) \)
3. **Compute** \( I(w) \) \( \triangleright \) Evaluate the goodness of the initial solution
4. **while** \( I(w) < I_{\text{max}} \) **do**
5. \( N := \text{SubsequentNoiseDetection}(w, \varepsilon) \) \( \triangleright \) Apply Theorem 6.1
to identify promising neighbors of \( w \)
6. \( w := \text{EvaluateCandidateSolution}(w, N) \) \( \triangleright \) Follow the steps
7-18 in Algorithm 1 to improve \( w \)
7. **if** \( I(w) \geq \sigma \) **then**
8. **Insert** \( w \) to \( S \)
9. **TYCOSLN**\((X_T, Y_T)\) \( \triangleright \) Restart TYCOSLN
10. **return** \( S \)

6.3 Setting the Correlation Threshold

6.3.1 Using normalized MI. Since MI is a measure of total
dependence between variables, its magnitude represents the
strength of the correlation. As the MI value is always non-negative,
its lower bound is 0. However, the MI’s upper bound varies and
thus, it is difficult to set an appropriate correlation threshold
using MI magnitude when data characteristics and their relations-
ships are unknown. To overcome this challenge, we propose
a robust method to set the correlation threshold based on the
normalized MI:

\[
0 \leq I_w = \frac{I_w}{H_w} \leq 1 \quad (18)
\]

where \( I_w \) is the MI and \( H_w \) is the entropy of the window \( w \).

In Eq. (18), the window entropy \( H_w \) represents the amount
of uncertainty contained in the window \( w \). Thus, \( I_w \) represents
the fraction of the window’s uncertainty reduced by the shared
information \( I_w \). The larger \( I_w \), the more information is shared
between the window’s variables, and thus the stronger correlation.
The normalized MI \( I_w \) is always scaled between [0, 1], and
thus provides an easier way for users to set the threshold \( \sigma \).

6.3.2 Using top-K filtering. Top-K maintains a list of \( K \) (K is
a predefined parameter) windows that have the highest MI up
to the current point. The top-K list represents the top correlated
time-series windows, and can be used to set the value of \( \sigma \). In
this top-K filtering approach, \( \sigma \) starts with the MI value of
the initial window \( w_0 \). As the search proceeds, the top-K list is
filled, and \( \sigma \) gets updated by the minimum MI value in the list.
Once the top-K list is full, it will get updated if there is a new window
that has MI greater than the current value of \( \sigma \). The element with
the least MI value in the top-K list will be replaced by this new
window, and \( \sigma \) is updated accordingly.

7 EFFICIENT MI COMPUTATION

In this section, we discuss the efficient MI computation (based
on Eq. (2)) in TYCOS. Due to space limitations, the discussion
will be brief and touch only important points.

Recall that while exploring its neighborhood, TYCOS might
visit the same data partition multiple times. For example, while
evaluating \( w_1^2 \) and \( w_2^2 \) in Fig. 5, TYCOS will repeatedly revisit \( w_i \) because \( w_1^1 \) and \( w_2^2 \) are extended from \( w_i \). To minimize the redundancy, we design an efficient MI computation method so that computation of overlapping data can be reused across windows. We observe that neighboring windows in each neighborhood \( N_c \) can differ from the current window \( w_i \) by only a small data partition \( w_i^{k \_IR} \), where \( w_i^{k \_IR} \) is either removed from or added to \( w_i \). For instance, in Fig. 5, \( w_1^1 \) differs from \( w_i \) by removing a \( w_i^{k \_IR} \) data partition from \( w_i \) whereas \( w_2^2 \) differs from \( w_i \) by adding a \( w_i^{k \_IR} \) data partition to \( w_i \). The removal of old data and the addition of new data can introduce different types of changes to the previous computation of \( w_i \). These changes can be either changing the \( k \)-nearest neighbors or changing the marginal counts \( n_x, n_y \) of existing points. To track those changes, we introduce the influenced region and influenced marginal region concepts for each data point.

Definition 7.1 (Influenced region (IR)) An IR of point \( p_1 = (x_1, y_1) \) is a square bounding box \( B_1 = (l_1, r_1, b_1, t_1) \), where \( l_1, r_1, b_1, t_1 \) are its left-, right-, bottom-, and top-most indices, respectively, and are computed as \( l_1 = x_1 - d, r_1 = x_1 + d, b_1 = y_1 - d, t_1 = y_1 + d \) where \( d = \max(d_s, d_y) \).

Definition 7.2 (Influenced marginal region (IMR)) The IMRs of point \( p_1 \) are the marginal regions located within the nearest distance \( d_s \) in each dimension.

Fig. 8 illustrates these concepts. The influenced region of \( p_1 \) is the square colored in green, and the influenced marginal regions are those with gray shade in either dimension.

Lemma 3. Given a window \( w_i \) and a data point \( p \in w_i \), a new point \( o \) inserted into \( w_i \) will become the new \( k \)-th neighbor of \( p \) if \( o \) is within IR of \( p \).

Lemma 4. Given a window \( w_i \) and a data point \( p \in w_i \), an existing point \( o \) deleted from \( w_i \) will change the \( k \) nearest points of \( p \) if \( o \) is within IR of \( p \).

Lemma 5. Given a window \( w_i \) and a data point \( p \in w_i \), a new point \( o \) inserted into \( w_i \) will increase the marginal count \( n_x \) (or \( n_y \)) of \( p \) if \( o \) is within IMR of \( p \).

Lemma 6. Given a window \( w_i \) and a data point \( p \in w_i \), an existing point \( o \) deleted from \( w_i \) will reduce the marginal count \( n_x \) (or \( n_y \)) of \( p \) if \( o \) is within IMR of \( p \).

Proof. Proofs of Lemmas 3, 4, 5, 6 are straightforward, thus omitted.

Lemmas 3, 4, 5, 6 display unique properties of IRs and IMRs. An IR maintains an area where any point \( p_j \) either falling into or being removed from this region will change the \( k \) nearest points of \( p_i \). In this case, a new \( k \)-nearest neighbors search is required for \( p_i \). Instead, an IMR maintains an area where any point \( p_j \) either falling into or being removed from it will change the marginal counts of \( p_i \). In this case, the marginalized neighbors of \( p_i \) have to be recounted.

Fig. 8 illustrates how changes are introduced and managed. For simplicity, we only discuss cases when new points are added into the previous computation. Changes introduced by removing points can be handled in a similar way. Assume that at time \( t_1 \), a new point \( p_8 \) is added to the current window and falls into the IR of \( p_1 \). The addition of \( p_8 \) changes the \( k \)-th nearest neighbor of \( p_1 \), thus triggers a new nearest neighbor search for \( p_1 \). At time \( t_2 \), a new point \( p_9 \) arrives and falls into the \( y \)-marginal influenced region of \( p_1 \), for which it will alter the marginal count \( n_y \) (but no new \( k \)-nearest neighbor search is required in this case). Similarly, a new point \( p_{10} \) will increase the marginal count \( n_x \). In these cases, only a recount of \( n_x \) or \( n_y \) is performed.

As the result of our efficient MI computation, for each window, only a minimum search region (containing new points) and a minimum update region (containing points affected by added and removed points) require additional computation. The rest is reused, and thus minimizing the computational cost.

8 EXPERIMENTAL EVALUATION

We evaluate the effectiveness and efficiency of TYCOS using both synthetic and real-world datasets. The effectiveness evaluates the method quantitatively by assessing the quality of extracted windows, while the efficiency evaluates the method quantitatively in terms of its performance and accuracy.

8.1 Baseline methods

Qualitative evaluation: TYCOS is compared against four baseline methods. The first baseline is a traditional correlation metric: Pearson Correlation Coefficient (PCC) [23]. The second is the Fast Subsequence Search (MASS) algorithm [25], often used for subsequences matching in time series. The third is MatrixProfile [31], considered to be the state of the art method for similarity join between time series. The final baseline is the Adaptive Mutual Information-based Correlation (AMIC) [17] framework that follows a top-down approach to search for multi-scale temporal correlations in big time series.

Quantitative evaluation: TYCOS runtime is compared against the Brute Force approach, and MatrixProfile (params are corresponding window lengths). Besides, different versions of TYCOS, i.e., LAHC-based TYCOS (TYCOS-L), TYCOS with noise theory applied (TYCOS NL), TYCOS with the efficient MI computation (TYCOS LMN), and TYCOS applying both noise theory and efficient MI computation (TYCOS LMN), are also compared against each other. The goal is to illustrate the effectiveness of the proposed theory and techniques. Note that we do not compare AMIC against TYCOS quantitatively because AMIC does not consider time delay correlations, and thus, it has a different search space. PCC and MASS are also not considered for quantitative evaluation because they do not have mechanisms to automatically search for correlated windows.

8.2 Parameter setting for TYCOS

TYCOS requires setting 5 parameters: correlation threshold \( \sigma \), noise threshold \( \epsilon \), minimum window size \( s_{min} \), maximum window size \( s_{max} \), and maximum time delay \( t_{max} \). Among these, \( \sigma, s_{min}, s_{max}, \) and \( t_{max} \) are user parameters, while \( \epsilon \) is a hyper parameter.

The value of \( \sigma \) determines the strength of extracted correlations. The larger the \( \sigma \), the stronger the correlations. In our experiments, we set the value of \( \sigma \) using the normalized MI (scaled between \([0, 1]\)) introduced in Section 6.3. On the other hand, the values of \( s_{min}, s_{max} \) and \( t_{max} \) are context dependent and is set based on domain knowledge. That is, given an application domain, it is usually intuitive how small/large a window could be and how long a time shift is possible. For example, when a user analyzes weather related data, he/she might decide that the longest duration of a weather event is two weeks, and thus set the size of \( s_{max} \) to two weeks. Similarly, a user can set \( t_{max} \) to 24 hours by assuming that weather events have impacts on other events only within a day duration. Table 2 lists the values of \( \sigma, s_{min}, s_{max}, \) and \( t_{max} \) we use in each dataset.

For the hyper parameter \( \epsilon \), we set \( \epsilon = \frac{\sigma}{2} \) in all experiments. This means that a window whose MI is less than 25% of the correlation threshold is considered unpromising to explore. The ratio \( \epsilon/\sigma = 0.25 \) is chosen based on empirical studies we conduct.
on different datasets, which consistently show that $\epsilon/\sigma \leq 0.25$ yields the best trade-off between accuracy and runtime gain. Section 8.5 shows this trade-off analysis, together with an analysis of the effects of $\sigma$, $s_{\text{max}}$ and $t_{\text{max}}$ on the performance of TYCOS.

### 8.3 Qualitative evaluation

#### A) Evaluation on synthetic datasets:
We generate synthetic datasets containing different types of relations, including both linear and non-linear, monotonic and non-monotonic, functional and non-functional functions. Then, we combine the generated relations into the same time series pair (the first time series is the values of $x$, the second time series is the values of $y = f(x)$). The individual relations are separated by independent data, and the time delays, $td=0$, 10, 50, 100, 150 (samples), are added between $x$ and $y$. Next, we apply TYCOS, and the baselines PCC, MASS, MatrixProfile and AMIC to the time series to verify whether the methods can detect the generated relations. A method detects a relation in a given pair of time series if it can locate a window $w$ where $(X_w,Y_w)$ corresponds to that relation. Table 1 shows the relations ($y = f(x)$ and $u$ is added noise) recognized by the tested methods (the $\checkmark$ sign denotes an identified relation, and the $\times$ sign denotes a unidentified relation). The plots of the generated relations can be found in [17].

We see that when there is no time delay ($td = 0$), TYCOS and AMIC can detect all types of relations, while PCC, MASS, and MatrixProfile cannot detect non-linear and non-functional relations, e.g., a circle relation. When there is time delay ($td \neq 0$), PCC, MASS and AMIC cannot detect any relations, while MatrixProfile can detect only linear relations, unlike TYCOS which can detect all the tested relations.

#### B) Evaluation on real-world datasets:
We evaluate TYCOS on two real-world data collections: smart energy [1] and smart city [2]. Using real-world applications, our goal is to make sense of extracted windows and learn insights from them. We describe the datasets, and the findings in the following.

The energy datasets [1]: measure energy usage from electrical devices in residential households in Maryland, USA during 07/2013-07/2014, and 02/2015-02/2016. There are 72 electrical plugs in total, and their consumptions are reported in minute and hour interval. We create pairwise time series from 72 plugs, and apply TYCOS and AMIC on each time series pair.

The smart city datasets: The NYC Open Data [2] contains more than 1,500 spatio-temporal datasets, providing rich information about NYC. For evaluation purposes, we consider two collections of data related to weather and transportation. Within transportation, we focus on the Collision dataset reporting the number of accidents in the city. The Weather dataset has 30 variables, recording weather condition in 5-minute and hour resolutions. The Collision dataset has 29 variables, recording incidents happened in minute resolution.

### Summary of the results:
On the energy datasets, TYCOS can extract correlations from more than 50 different time series pairs, while AMIC extracts fewer windows than TYCOS, and omits any correlations that have time delay. On smart city datasets, TYCOS is able to find correlations that could not be confirmed in [17] by AMIC. Due to space limitations, we cannot discuss all of them, but instead just show a few extracted correlations in Table 3 to illustrate our observations. In each column, the first number is the number of extracted windows, the second number is the time delay range, and the $\times$ sign denotes no windows can be extracted.

### Table 3: Extracted correlations (h: hour, m: minute)

<table>
<thead>
<tr>
<th>Correlations</th>
<th>TYCOS</th>
<th>AMIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>(C1) Kitchen vs. Dish Washer</td>
<td>80</td>
<td>25</td>
</tr>
<tr>
<td>(C2) Kitchen vs. Microwave</td>
<td>21</td>
<td>5</td>
</tr>
<tr>
<td>(C3) Clothes Washer vs. Dryer</td>
<td>9.9</td>
<td>1</td>
</tr>
<tr>
<td>(C4) Bathroom Light vs. Kitchen Light</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>(C5) Kitchen Light vs. Microwave</td>
<td>1.1</td>
<td>4.9</td>
</tr>
<tr>
<td>(C6) Children Room Light vs. Living Room Light</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>(C7) Precipitation vs. Collisions</td>
<td>28</td>
<td>0.5-2</td>
</tr>
<tr>
<td>(C8) Wind Speed vs. Collisions</td>
<td>23</td>
<td>0.25-16</td>
</tr>
<tr>
<td>(C9) Precipitation vs. Pedestrian Injured</td>
<td>16</td>
<td>0.5-2</td>
</tr>
<tr>
<td>(C10) Wind Speed vs. Motorist Killed</td>
<td>12</td>
<td>0.25-16</td>
</tr>
</tbody>
</table>

**Interpretation of extracted windows:** We interpret some of the correlations in Table 3 by comparing with the findings of [7, 17], and/or by plotting the data of extracted windows. Here, C1 presents a correlation between the energy usage of the kitchen and of the dish washer, with the time shift ranging from 0 to 4 hours. The extracted windows indicate frequent activities of kitchen from 16.00 to 19.00, and of dish washer from 21.00 to 23.00. C4 presents a correlation between the light upstairs in the bathroom, and the light downstairs in the kitchen, with an average time shift from 1 to 5 minutes. The correlation occurs frequently from 6.00 to 7.00. This pattern might hint that, either more than one person are living together so that when one is in the bathroom, the other goes to the kitchen; or that the same person wakes up in the early morning, goes to the bathroom and then comes to the kitchen. Interestingly, C5 can help provide extra information for C4. A correlation between the kitchen light and the microwave is identified, with a time shift between two devices is from 0 to 2 minutes, indicating the person might come to the kitchen to prepare breakfast. On smart city datasets, C7 and C8 present correlations between the increase of precipitation/wind speed, and the number of collisions, with a time shift from 0.25 to 2 hours. In [17], AMIC could not confirm C7 and C8, because it does not consider the time delay between time series, and thus fail to capture correlations that are shifted in time. Furthermore, we found that precipitation has stronger impact on pedestrians.
8.4 Quantitative evaluation

TYCOS performance is evaluated in terms of its runtime and accuracy. TYCOS is implemented in C++, and the experiments are run on a standard PC that has 2.7 GHz processor, 16 GB of RAM, and 512 GB of SSD.

A) Runtime evaluation: TYCOS runtime is evaluated by comparing its 4 different versions: TYCOSL, TYCOSLN, TYCOSLM, TYCOSLMN, and the Brute Force and MatrixProfile baselines. First, different TYCOS versions are compared against each other. The results on both synthetic and real-world data are shown in Fig. 9. The synthetic datasets, Synthetic 1, Synthetic 2, and Synthetic 3, are created by combining multiple relations from Table 1 into one time series pair. From Fig. 9 where the y-axis is in log scale, it can be seen that TYCOSLMN achieves the best performance among all versions. Its speedup w.r.t. TYCOSL ranges from 10 to 150 depending on data sizes. The average speedup is 20 on synthetic data, and 60 on real-world data. Furthermore, the noise theory and the efficient MI computation technique result in different speedups depending on data (there are situations where the noise theory is more efficient, and vice versa). The average speedup is 39 for the noise theory, and 32 for the efficient MI computation. However, applying both always yields better speedups than applying either of them.

Next, TYCOS with the best performance, TYCOSLMN, is compared against Brute Force and MatrixProfile. The results are shown in Fig. 10 (note log scale in the y-axis). We can see that TYCOSLMN can achieve an average speedup of more than 3 orders of magnitude over Brute Force, and of more than 2 orders of magnitude over MatrixProfile, both of which are, however, exact.

B) Accuracy evaluation: To evaluate the accuracy of TYCOS, we compare the similarity of windows extracted from 3 versions: Brute Force, TYCOSL, and TYCOSLN. Note that the efficient MI computation technique does not change the accuracy of TYCOSL, thus, TYCOSLM and TYCOSLMN are not considered in this evaluation. Moreover, two windows are considered to be similar if they cover a similar range of indices. The comparison between Brute Force and TYCOSLN evaluates how accurate the LAHC approach on the TYCOS problem is, while the comparison between TYCOSL and TYCOSLM validates the accuracy of the noise theory. Since Brute Force generates overlapped windows, the generated windows are aggregated and the overlapped windows are combined together. The same synthetic and real-world datasets as when evaluating the runtime are used in these experiments.

Table 4 shows the average accuracy of TYCOSL w.r.t. Brute Force, and of TYCOSLN w.r.t. TYCOSL. Depending on the data sizes, TYCOSLN extracts from 88% to 98% similar windows compared to Brute Force, while TYCOSLMN extracts windows that are from 90% to 100% similar to TYCOSL.

The quantitative evaluation proves that our proposed theory and technique are very effective in improving the search performance. They help achieve an average speedup of more than 3 orders of magnitude compared to the Brute Force method, while maintaining highly accurate results.

8.5 Effects of Parameters

We examine how the major parameters: $\varepsilon$, $\sigma$, $s_{\max}$, and $t_{\max}$ affect the performance of TYCOS. We do not consider $s_{\min}$ in this experiment because $s_{\min}$ has minimal impact on TYCOS results.

A) Noise threshold $\varepsilon$: First, we examine how different values of $\varepsilon$ affect the accuracy and runtime, using both synthetic and real-world data in Fig. 11. We can see, as the ratio $\varepsilon/\sigma$ increases, the runtime gain increases (Fig. 11b), but the error rate also increases (Fig. 11a, error rate is measured by the number of missing windows). This result is intuitive because as the ratio $\varepsilon/\sigma$ increases, more of the TYCOS search space is pruned, leading to higher speedup and larger errors. Next, we perform a trade-off analysis between accuracy and runtime gain as a means for choosing a proper value of the noise threshold $\varepsilon$. In Fig. 12, the accuracy and the runtime gain of each tested dataset are plotted together, with the ratio $\varepsilon/\sigma$ on the x-axis. On the two tested datasets, i.e., energy and smart city datasets, we found that, when $\varepsilon/\sigma \in [0.05, 0.3]$, TYCOSLN maintains an error rate less than 5%, while reducing the runtime up to 50%, compared to TYCOSL. Thus, our experimental setting $\varepsilon = 0.25$ proved to be effective and robust. This threshold can be adjusted according to user’s preference for accuracy.
B) Correlation Threshold \( \sigma \): We vary the values of \( \sigma \) to examine its effect, shown in Fig. 15a. We observe that, the correlations are stronger as \( \sigma \) increases, and thus, fewer windows are extracted. However, the runtime also increases because larger neighborhoods need to be explored to find strong correlations. For example, only 80 windows are extracted compared to 681 windows when \( \sigma \) increases from 0.2 to 0.6, while the runtime increases from 115 to 573 seconds.

C) Window Size \( s_{\text{max}} \) and Time Delay \( t_{\text{max}} \): We examine how \( s_{\text{max}} \) and \( t_{\text{max}} \) affect TYCOS. We found that, although \( s_{\text{max}} \) and \( t_{\text{max}} \) are context-dependent, the algorithm will converge after the two parameters reach certain values. When the convergence occurs, TYCOS extracts the same set of windows, while maintaining a similar runtime for \( t_{\text{max}} \), but an increasing runtime for \( s_{\text{max}} \). Fig. 13b and Fig. 13c illustrate this evaluation. Here, using the (Snow, Collision) datasets, TYCOS converges at \( s_{\text{max}} = 250 \) and \( t_{\text{max}} = 60 \), with 276 windows extracted when the convergence occurs. After the convergence, the runtime continues increasing as \( s_{\text{max}} \) goes beyond the value 250, while keeping similar values as \( t_{\text{max}} \) goes more than 60.

9 CONCLUSION AND FUTURE WORK

To our knowledge, TYCOS is the first comprehensive solution for discovering correlations across windows when integration of TYCOS and LAHC for multi-scale time delay search. (2) the novel MI-based theory for noise identification, functional and non-functional ones. Our major contributions are:

- TYCOS has the ability to extract all types of correlation relations, including both linear and non-linear, monotonic and non-monotonic, functional and non-functional ones. Our major contributions are:
- The ability to detect various types of relations in synthetic datasets, and identify significant and interesting correlations in real-world datasets. The proposed noise theory and MI computation technique are also proved to be effective and improve the search performance by 2 to 3 orders of magnitude compared to the baselines. In future work, TYCOS can be extended to capture correlations across spatial dimensions. The result of this work can also provide a foundation for deeper data analysis, such as performing mining or inferring causal effects from the extracted correlations.

ACKNOWLEDGMENT

This work has been partially supported by the DICPYS project funded by Innovation Fund Denmark, the GoFLEX project funded by the Horizon 2020 program, the SEMIOTIC project funded by Independent Research Fund Denmark, and the AAU International Postdoctoral Program.

REFERENCES