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

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 COrrelation 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 noise 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 autonomic 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 raises 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 un-correlated and negatively correlated time series, both of which may 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 stocks is visible only a few hours later. Given a heterogeneous set of time series data, there is a need to identify not only which datasets are correlated, but also when the correlations occur, and at what time delay.

Although correlation analysis has been researched extensively, current techniques are limited either in the type of detected relations, i.e., only linear ones, or the temporal scale and time delay in which they deal with. Most of the correlation works do not consider adaptive temporal scales as they assume correlations only exist for a fixed time period, e.g., [29], or ignore the time delay between variables of interest. There is no existing work that offers a holistic solution for searching window-based correlations, considering both multiple temporal scales and time delay, in modern big time series data.

This paper aims to address those challenges and limitations by introducing the Time delaY COrrelation Search (TYCOS) approach, making the following novel contributions: (1) We propose the first, to our knowledge, comprehensive solution for the multi-scale time delay correlation search problem that extracts significant correlations from big time series. (2) TYCOS is based on the concept of mutual information from information theory, giving it a strong theoretical foundation and the ability to discover various types of correlation relations, including linear and non-linear, monotonic and non-monotonic, functional and non-functional. (3) By combining the well-known Late Acceptance Hill Climbing (LAHC) search method with a window-based approach, TYCOS can automatically discover time delay correlations at multiple temporal scales, without requiring user inputs to specify the window sizes or the delay. (4) Based on mutual information properties, we propose a novel theory to identify noise in the data, enabling efficient pruning of non-interesting time intervals from the search, thus significantly reducing the search space and improving the search speed. (5) TYCOS is designed with efficient data structures to reuse the MI computation across a large number of windows, thus minimizing the computation redundancy. Moreover, TYCOS is scalable since it is designed in an efficient bottom-up fashion, making the method memory efficient and suitable for big datasets. And finally (6), we perform a comprehensive experimental evaluation using synthetic and real-world datasets from the energy and smart city domains, which shows that TYCOS is able to find interesting and important correlations among time series with high accuracy, can scale to large datasets, and achieves an average speedup of 2 to 3 orders of magnitude compared to the baselines.

2 RELATED WORK

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 had 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) $p(x)$, $p(y)$, and the joint distribution $p(x, y)$, the MI between X and Y is defined as

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \tag{1}$$

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 accuracy [22]; (2) The method uses $k-$nearest neighbor approximation and thus is data efficient, adaptive and has minimal bias [20]. These reasons make the *KSG* method particularly suitable for discovering temporal correlations in big and heterogeneous time series where the dependencies between different time series can be complex and might occur at multiple time scales.

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 within d_k. This leads to the computation of MI between two variables X and Y as [20]:

$$I(X;Y) = \psi(k) - 1/k - \langle \psi(n_x) + \psi(n_y) \rangle + \psi(n) \quad (2)$$

where ψ 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 $\langle \cdot \rangle$ is the average function. The digamma function ψ is a monotonically increasing function. Thus, the larger n_x and n_y (i.e., more data points fall within 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_h to maintain a history of the most recently accepted solutions, and use L_h to justify the goodness of a candidate solution.

4 PROBLEM FORMULATION

Definition 4.1 *(Time series)* A time series $X_T = \{x_1, x_2, ..., x_n\}$ is a sequence of data values that measures the same phenomenon during an (observation) time 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 $X_w = \{x_{t_s}, ..., x_{t_e}\} \subseteq X_T$.

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 $(X_T, Y_T) = (\{x_1, x_2, ..., x_n\}, \{y_1, y_2, ..., y_n\})$ 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 $w_X = [t_{x_s}, t_{x_e}]$, $w_Y = [t_{y_s}, t_{y_e}]$ 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 $(w_X, w_Y) = ([t_{x_s}, t_{x_e}], [t_{y_s}, t_{y_e}])$ records the events of X_T from $[t_{x_s}, t_{x_e}]$, and of Y_T from $[t_{y_s}, t_{y_e}]$, and forms a pair of (sub) time series $(X_w, Y_w) = (\{x_{t_{x_s}}, ..., x_{t_{x_e}}\}, \{y_{t_{y_s}}, ..., y_{t_{y_e}}\}) \subseteq (X_T, Y_T)$.

Definition 4.5 *(Time delay window of a time series pair)* Let $(w_X, w_Y) = ([t_{x_s}, t_{x_e}], [t_{y_s}, t_{y_e}])$ be a pair of time windows like in Definition 4.4, and τ be an integer. The pair (w_X, w_Y) is called a time delay window of (X_T, Y_T) with the delay τ if $t_{y_s} - t_{x_s} = \tau$, and is denoted as $w_{X,Y+\tau} = ([t_s, t_e], \tau)$, where $t_s = t_{x_s}$ and $t_e = t_{x_e}$ are the start time and the end time of $w_{X,Y+\tau}$ on X_T, and τ is the time delay of w_Y w.r.t. w_X.

The window $w_{X,Y+\tau} = ([t_s, t_e], \tau)$ in Definition 4.5 defines a one-to-one mapping $f\colon w_X \mapsto_\tau w_Y$ 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_e, and the time delay τ. The size of $w_{X,Y+\tau}$ equals to the size of w_X and w_Y, i.e., $|w_{X,Y}| = |w_X| = |w_Y|$.

A time delay window represents a shift (also called a "delay" or "lag") in time between two time series X_T and Y_T, and the value of τ indicates the shifted time units. Since τ can be equal to 0, or positive, or negative, the window $w_{X,Y+\tau} = ([t_s, t_e], \tau)$ is generalized 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_Y is delayed τ time units from w_X (or events in w_Y occur τ time units after events in w_X). Similarly, if $\tau < 0$, w_X is delayed τ time units from w_Y. For brevity, in this paper, the two terms *time delay window* and *window* are used interchangeably.

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, $w_1 = ([t_{s_1}, t_{e_1}], \tau_1 = 0])$ has no time delay, thus starts and ends at the same time on X_T and Y_T. Instead, the window $w_2 = ([t_{s_2}, t_{e_2}], \tau_2 > 0])$ has a time delay $\tau_2 > 0$, thus Y_T is shifted from X_T. The window $w_3 = ([t_{s_3}, t_{e_3}], \tau_3 < 0])$ has $\tau_3 < 0$, thus X_T is shifted from Y_T, similarly for w_4.

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:

$$I_{w_{X,Y+\tau}} = I(X_w; Y_w) = \psi(k) - \frac{1}{k} - \frac{1}{m} \sum_{\substack{x_i \in X_w \\ y_j \in Y_w}} [\psi(n_{x_i}) + \psi(n_{y_j})] + \psi(m) \quad (3)$$

where m is the size of $w_{X,Y+\tau}$, and n_{x_i} and n_{y_j} are the number of data points falling within the k^{th}-nearest distances in each dimension d_x and d_y of point $(x_i, y_j) \in (X_w, Y_w)$.

Fig. 2 illustrates how to compute the MI of a window using *KSG* estimation. Consider a window $w_{X,Y+\tau}$ contains 7 data points

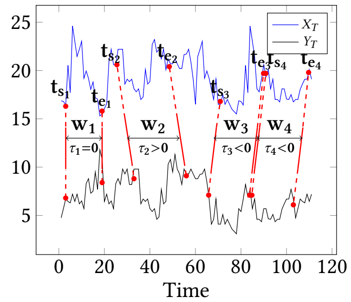


Figure 1: Time windows

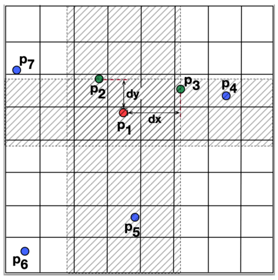


Figure 2: MI computation

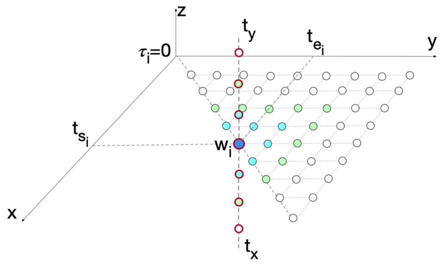


Figure 3: Search space of TYCOS

$\{p_1=(x_1, y_1), ..., p_7=(x_7, y_7)\}$, with their positions projected into a two dimensional grid as in Fig. 2. Without loss of generality, we assume $\tau = 0$ (events in X_w and Y_w occur at the same time step), the nearest neighbor parameter $k = 2$, and the distance metric between neighbors is the *maximum norm*[1]. Under this setting, the 2 nearest neighbors of p_1 are p_2 and p_3 (in green), and the 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_{x_1}=3 (including p_2, p_3, p_5), and n_{y_1}=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_{x_i}, n_{y_i} are inserted into Eq. 3 to compute the MI of $w_{X,Y+\tau}$.

Definition 4.7 *(Correlated time delay window)* Let $w_{X,Y+\tau}$ be a *time delay window* of (X_T, Y_T), and $I_{w_{X,Y+\tau}}$ be the MI of $w_{X,Y+\tau}$. The two time series X_T and Y_T are said to be correlated within $w_{X,Y+\tau}$ iff $I_{w_{X,Y+\tau}} \geq \sigma$ where $\sigma > 0$ is a pre-defined correlation threshold.

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 $w_{X,Y+\tau}$ be a time delay window of (X_T, Y_T). Then TYCOS aims to find a set S of $w_{X,Y+\tau}$ such that $s_{\min} \leq |w_{X,Y+\tau}| \leq s_{\max} \wedge \tau \leq td_{\max} \wedge I_{w_{X,Y+\tau}} \geq \sigma \wedge \forall w_i, w_j \in S : w_i \nsubseteq w_j \wedge w_j \nsubseteq w_i$, where $|w_{X,Y+\tau}|$ denotes the size of $w_{X,Y+\tau}$, $s_{\min}$ and $s_{\max}$ are the minimum and maximum sizes that a window can have, $td_{\max}$ is the maximum time delay, and σ is the pre-defined correlation threshold.*

The goal of TYCOS is to find a set S of non-overlapping time delay windows that respect size and time delay constraints, and have their MI satisfying the pre-defined correlation threshold. As the size of each window is restricted in the range $[s_{\min}, s_{\max}]$, this implies that if correlations exist in the pair (X_T, Y_T), they will last at least for length $s_{\min}$, and at most for length $s_{\max}$. This assumption is meaningful especially when working with real datasets. For example, when searching for weather-related correlations, one could assume that correlations can only last for at most *several* weeks which correspond to the longest duration of a weather event. Furthermore, the time delay of a window is also assumed to be bounded by a maximum value $td_{\max}$ that represents the longest shifting duration between two time series. The value of $td_{\max}$ can be used to prevent spurious correlations. For example, a heavy rain cannot have an impact on the number of injured pedestrians one year later. Setting $td_{\max}$ value, for now, will rely on the expert's domain knowledge.

[1] L_∞: $d(p_i, p_j) = \| (d_x, d_y) \|_{\max} = \max(\| x_i - x_j \|, \| y_i - y_j \|)$

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 τ of a window. Each point in this 3−dimensional grid represents a window w_i identified by its start time index t_{s_i}, end time index t_{e_i}, time delay τ_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, $td_{\max}$ 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 $w_0 = ([t_{s_0}, t_{e_0}], \tau_0)$ at the minimum size $s_{\min}$ and the initial time delay $\tau_0 = 0$. 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 $(2 * td_{\max})$ times (corresponding to *negative* and *positive* values of τ), creating $(2 * td_{\max})$ possible time delay windows. Finally, there are $(n - s_{\min} + 1)$ possible start indices t_{s_i}. Thus, the total number of feasible windows of TYCOS is:

$$(n - s_{\min} + 1) * (s_{\max} - s_{\min} + 1) * 2 * td_{\max} \sim n^3 \tag{4}$$

if $s_{\max} \to n \wedge td_{\max} \to n \wedge s_{\min} \ll n$. □

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 $O(n^3 m^2)$.*

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.

On the other hand, the MI is computed using the *KSG* estimator, in which the most expensive operator is the k-nearest neighbor (*kNN*) search. Therefore, the complexity of MI computation depends on the complexity of *kNN* search. Consider a window w_i of size m. A basic *kNN* algorithm applied to w_i will require $O(kdm)$ to find k nearest neighbors for one sample (d is the data dimensionality), and thus $O(kdm^2) \sim O(m^2)$ (if k and d are significantly smaller than m) for all samples in w_i [12]. Hence, the worst-case time complexity of a Brute Force search for TYCOS is $O(n^3m^2)$. 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 $O(kdm \log m) \sim O(m \log m)$, and thus, the expected-case Brute Force complexity is $O(n^3 m \log m)$. □

5.2 TYCOS$_{\mathbf{LAHC}}$: 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, $s_{\max} = 400$, $s_{\min} = 20$, and $td_{\max} = 20$ 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 σ (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 TYCOS$_{\mathbf{LAHC}}$ (or TYCOS$_{\mathbf{L}}$ in short), as a *maximization* problem. Specifically, the target function of TYCOS$_{\mathbf{L}}$ is a *maximize* function, and our goal is to find windows where their MIs are locally maximal values that satisfy σ.

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 $w_i = ([t_{s_i}, t_{e_i}], \tau_i)$ is the window where the search is currently at. Starting from w_i, if TYCOS$_{\mathbf{L}}$ follows a rightwards trajectory on the $y-$axis, it moves the end time t_{e_i} 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_{e_i} backward in time, thus reducing the window size. Similarly, moving along the $x-$axis, TYCOS$_{\mathbf{L}}$ can reduce or increase the start time t_{s_i}, therefore, extending or narrowing the size of w_i accordingly. On the $z-$axis, following the t_x direction, TYCOS$_{\mathbf{L}}$ increases the shifting time of X_T w.r.t. Y_T. Following the t_y direction, TYCOS$_{\mathbf{L}}$ 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, TYCOS$_{\mathbf{L}}$ 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 *(δ-neighbor)* Let $w = ([t_s, t_e], \tau)$ be a window of (X_T, Y_T), and assume (X_T, Y_T) has length n. A window $w' = ([t'_s, t'_e], \tau')$ is a *δ-neighbor* of w if $t'_s = t_s \pm \delta \vee t'_e = t_e \pm \delta \vee \tau' = \tau \pm \delta$, where δ is a pre-defined moving step such that $1 \leq \delta \leq n \wedge s_{\min} \leq \left|w'\right| \leq s_{\max} \wedge \tau' \leq td_{\max}$.

A δ-neighbor window w' has at least one of its indices (i.e., $t_{s'}$, or $t_{e'}$, or τ') differing a δ step from the indices of w.

Definition 5.2 *(δ-neighborhood)* Let $w = ([t_s, t_e], \tau)$ be a window of (X_T, Y_T). A *δ-neighborhood* of w, denoted as N_δ, is formed by all *δ-neighbors* $w' = ([t'_s, t'_e], \tau')$ 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_1, is the area formed by the 26 windows in blue color w_i^1 where $i = 1, ..., 26$. Each window in this neighborhood differs from w_i by *one* δ 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_2, is the 50 windows in green color area. Each $\delta-neighborhood$ forms an area where TYCOS$_{\mathbf{L}}$ will iteratively look for potential candidates to improve w_i.

b) TYCOS$_{\mathbf{L}}$ algorithm. We provide the outline of TYCOS$_{\mathbf{L}}$ in Algorithm 1, and explain it in the following.

Consider a time series pair (X_T, Y_T), and let I_w be the target function to be maximized. To improve I_w, TYCOS$_{\mathbf{L}}$ will start with an initial feasible solution, and explores its neighborhood to look for better solutions. Let $w = w_0$ where$|w_0| = s_{\min} \wedge \tau_0 = 0$ 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, TYCOS$_{\mathbf{L}}$ will first explore its nearest neighborhood N_1, and search for a better solution than w_0 in this area. To do that, it creates all δ_1-neighbors of w_0 to form N_1. Then for each $w' \in N_1$, it computes $I(w')$ 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: $I_{bestnb} > I_w$ or $I_{bestnb} > I_{w_h}$ where $w_h \in L_h$, 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 $I_{bestnb} \leq I_{w_h}$, 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_h to justify a potential candidate solution, the newly selected solution *bestnb* might be better than w_h, 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 TYCOS$_{\mathbf{L}}$ stops, the time series pair might not be scanned entirely. In that case, TYCOS$_{\mathbf{L}}$ restarts again on the remaining part of the data, looking

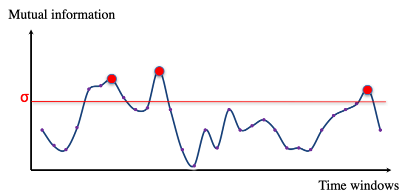


Figure 4: MI fluctuation

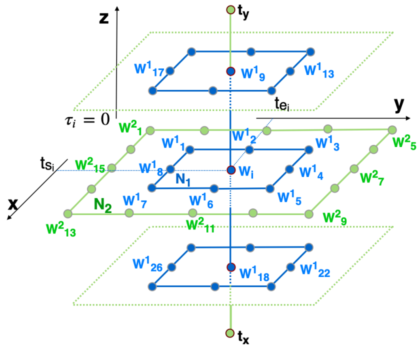


Figure 5: Explore neighborhood

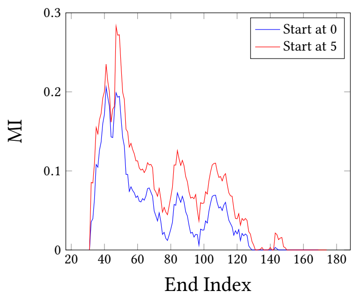


Figure 6: Changes of MI

for new local optimal solutions, until the entire time series are searched (line 21).

Stopping conditions: Ideally, TYCOS$_\mathbf{L}$ 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 $T_{maxIdle}$ 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$_\mathbf{L}$ maintains a history L_h of the most recently accepted solutions and uses it to justify the goodness of a potential candidate. In our implementation, TYCOS$_\mathbf{L}$ follows the *random* policy when selecting and updating an item in the history (line 9 and 16 − 18).

Algorithm 1 TYCOS$_\mathbf{L}$: LAHC for TYCOS

Input: (X_T, Y_T): pair of time series
Params: σ, ε, $s_{\min}$, $s_{\max}$, $td_{\max}$
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 $w := w_0$ with $|w_0| = s_{\min} \wedge \tau_0 = 0$
3: Compute $I(w_0)$ ▹ Evaluate the goodness of w_0
4: **while** $t_{idle} < T_{maxIdle}$ **do**
5: $N := Neighbors(w)$ ▹ Identify the neighbors of w
6: **for** $w' \in N$ **do**
7: Compute $I(w')$ ▹ Evaluate the goodness of w'
8: $bestnb := BestNeighbor(N)$ ▹ Select best neighbor in N
9: $w_h := random.get(L_h)$ ▹ Randomly select from L_h
10: **if** $I_{bestnb} > I_{w_h}$ **or** $I_{bestnb} > I_w$ **then**
11: $w := bestnb$ ▹ Accept the candidate
12: $t_{idle} := 0$ ▹ Reset the idle time
13: **else**
14: $w := w$ ▹ Reject the candidate
15: $t_{idle} := t_{idle} + 1$ ▹ Increase the idle time
16: **if** $I_w > I_{w_h}$ **then** ▹ Update the history list
17: $w_h := w$
18: $I_{w_h} := I_w$
19: **if** $I_w \geq \sigma$ **then**
20: Insert w to S
21: $TYCOS_\mathbf{L}(X'_T, Y'_T)$ ▹ Restart TYCOS$_\mathbf{L}$
22: **return** S

6 NOVEL NOISE THEORY TO IMPROVE TYCOS

6.1 Noise Identification

When TYCOS$_\mathbf{L}$ 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, TYCOS$_\mathbf{L}$ 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 partition in 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_i (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, $w^1_4 \in N_1$ is expanded from w_i by extending its end index by a δ_1 step, while $w^2_7 \in N_2$ is an extension of w^1_4 by enlarging w_i's end index a δ_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 times 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 θ 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 θ and is written as $Z = X \odot_\theta U$.

Theorem 6.1. *Let X, Y, U, V be discrete random variables and $p_X(x), p_Y(y), p_U(u)$, and $p_V(v)$ be their corresponding p.m.fs. Let*

$Z = X \odot_\theta U$ and $W = Y \odot_\eta V$ where $\odot$ denotes the mixture of two variables. Assume that, except for X and Y, other variables are pair-wise independent, i.e., $(U \perp V) \wedge (X \perp U) \wedge (X \perp V) \wedge (Y \perp U) \wedge (Y \perp V)$. Then $I(X;Y) \geq I(Z;W)$.

Proof. Z and W are the two mixed variables: $Z = X \odot_\theta U$ and $W = Y \odot_\eta V$. 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:

$$p_Z(x) = P(Z = X)p_X(x) = \theta p_X(x) \tag{5}$$

$$p_Z(u) = P(Z = U)p_U(u) = (1-\theta)p_U(u) \tag{6}$$

Similarly, for $y \sim p_Y(y)$ and $v \sim p_V(v)$, we have:

$$p_W(y) = P(W = Y)p_Y(y) = \eta p_Y(y) \tag{7}$$

$$p_W(v) = P(W = V)p_V(v) = (1-\eta)p_V(v) \tag{8}$$

Then, we can write the following joint probabilities:

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

$$p_{Z,W}(x,v) = \theta(1-\eta)p_{X,V}(x,v) \tag{10}$$

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

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

We have the MI between X and Y as

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

And the MI between Z and W as

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

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

$$\begin{aligned} I(Z;W) = &\sum_{w\in\mathcal{R}_Y}\sum_{z\in\mathcal{R}_X} p_{Z,W}(x,y)\log\frac{p_{Z,W}(x,y)}{p_Z(x)p_W(y)} \\ &+ \sum_{w\in\mathcal{R}_Y}\sum_{z\in\mathcal{R}_U} p_{Z,W}(u,y)\log\frac{p_{Z,W}(u,y)}{p_Z(u)p_W(y)} \\ &+ \sum_{w\in\mathcal{R}_V}\sum_{z\in\mathcal{R}_X} p_{Z,W}(x,v)\log\frac{p_{Z,W}(x,v)}{p_Z(x)p_W(v)} \\ &+ \sum_{w\in\mathcal{R}_V}\sum_{z\in\mathcal{R}_U} p_{Z,W}(u,v)\log\frac{p_{Z,W}(u,v)}{p_Z(u)p_W(v)} \\ &= \sum_{y\in\mathcal{R}_Y}\sum_{x\in\mathcal{R}_X} \theta\eta p_{X,Y}(x,y)\log\frac{\theta\eta p_{X,Y}(x,y)}{\theta p_X(x)\eta p_Y(y)} \\ &+ \sum_{y\in\mathcal{R}_Y}\sum_{u\in\mathcal{R}_U} (1-\theta)\eta p_{U,Y}(u,y)\log\frac{(1-\theta)\eta p_{U,Y}(u,y)}{(1-\theta)p_U(u)\eta p_Y(y)} \\ &+ \sum_{v\in\mathcal{R}_V}\sum_{x\in\mathcal{R}_X} \theta(1-\eta)p_{X,V}(x,v)\log\frac{\theta(1-\eta)p_{X,V}(x,v)}{\theta p_X(x)(1-\eta)p_V(v)} \\ &+ \sum_{v\in\mathcal{R}_V}\sum_{u\in\mathcal{R}_U} (1-\theta)(1-\eta)p_{U,V}(u,v)\log\frac{(1-\theta)(1-\eta)p_{U,V}(u,v)}{(1-\theta)p_U(u)(1-\eta)p_V(v)} \end{aligned} \tag{15}$$

Eq. (15) can be rewritten as

$$\begin{aligned} I(Z;W) = &\ \theta\eta I(X;Y) + (1-\theta)\eta I(U;Y) \\ &+ \theta(1-\eta)I(X;V) + (1-\theta)(1-\eta)I(U;V) \end{aligned} \tag{16}$$

Since we assume

$$(U \perp V) \wedge (X \perp U) \wedge (X \perp V) \wedge (Y \perp U) \wedge (Y \perp V)$$

This leads to

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

Thus, Eq. (16) becomes

$$I(Z;W) = \theta\eta I(X;Y) \tag{17}$$

where $\theta \leq 1$ and $\eta \leq 1$, which leads to

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

□

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 $w_{X,Y+\tau} = ([t_s, t_e], \tau)$ and $w'_{X,Y+\tau'} = ([t_{s'}, t_{e'}], \tau')$ be the two time delay windows of (X_T, Y_T). Then $w_{X,Y+\tau}$ and $w'_{X,Y+\tau'}$ are consecutive iff $t_{s'} = t_e + 1 \wedge \tau = \tau'$.

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 $w_{X,Y+\tau} = ([t_s, t_e], \tau)$ and $w'_{X,Y+\tau'} = ([t_{s'}, t_{e'}], \tau')$ be two consecutive windows of (X_T, Y_T). The concatenation between $w_{X,Y+\tau}$ and $w'_{X,Y+\tau'}$ is defined as: $w''_{X,Y+\tau} = w_{X,Y+\tau} \odot w'_{X,Y+\tau'} = ([t_s, t'_e], \tau)$. 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 ε $(0 \leq \varepsilon < \sigma)$ 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 $I_{w'_{X,Y+\tau'}} < \varepsilon \wedge I_{w''_{X,Y+\tau}} < I_{w_{X,Y+\tau}}$.

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 $\text{TYCOS}_\mathbf{L}$. We name $\text{TYCOS}_\mathbf{L}$ with noise theory applied as $\text{TYCOS}_\mathbf{LN}$.

6.2.1 Initial noise pruning. Previously, we said that $\text{TYCOS}_\mathbf{L}$ can start out at the beginning of, or at an arbitrary location in the time series. This, however, can lead the search to an unpromising exploration area. For example, if the search starts out at the valleys in Fig. 4, it might take longer time to reach the top of the hill than if it starts somewhere on the edges. To avoid the

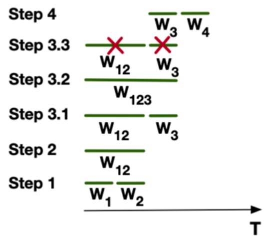


Figure 7: Initial window

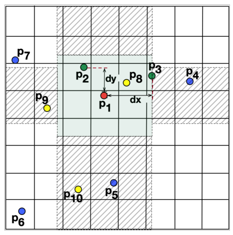


Figure 8: Efficient MI computation

"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 $s_{\min}$ 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*, initially the search starts with two minimal consecutive 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 ε, 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 $I_{w_{123}}$).

Assume that $I_{w_3} < \varepsilon$, and that by combining w_3 to w_{12}, it reduces the MI $I_{w_{12}}$, i.e., $I_{w_{123}} < I_{w_{12}} < \varepsilon$. According to Theorem 6.1, we can conclude that w_3 is noise w.r.t. w_{12}. Thus, the combination w_{123} 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_{1234} 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 > ε. Once the starting point is determined, TYCOS$_{\mathbf{LN}}$ 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_i is the current window and w_4^1, w_7^2 are its neighbors when moving along the $y-$axis. In the first exploration, the neighbor w_4^1 is considered. Since w_4^1 is created by extending the end index of w_i by a δ_1-step, we have: $w_4^1 = w_i \odot w_{\delta_1}$ where w_{δ_1} is the extension to be concatenated with w_i. Assume that by applying our noise theory to w_i, w_{δ_1}, and w_4^1, we conclude that w_{δ_1} is noise w.r.t. w_i. In this case, it is not promising to further explore the neighborhoods of w_i along the $y-$axis in that direction. In the next exploration, TYCOS$_{\mathbf{LN}}$ will omit w_7^2, as well as the entire forward direction along the $y-$axis.

Ensuring the completeness of TYCOS$_{\mathbf{LN}}$: When TYCOS$_{\mathbf{LN}}$ 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, TYCOS$_{\mathbf{LN}}$ 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 TYCOS$_{\mathbf{LN}}$: Apply noise theory to TYCOS$_{\mathbf{L}}$

Input: (X_T, Y_T): pair of time series
Params: σ, ε, $s_{\min}$, $s_{\max}$, $td_{\max}$
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 $w := InitialNoisePruning((X_T, Y_T), \varepsilon)$
3: *Compute* $I(w)$ ▹ Evaluate the goodness of the initial solution
4: **while** $t_{idle} < T_{maxIdle}$ **do**
5: $N := SubsequentNoiseDetection(w, \tau)$ ▹ Apply Theorem 6.1 to identify promising neighbors of w
6: $w := EvaluateCandidateSolution(w, N)$ ▹ Follow the steps 8-18 in Algorithm 1 to improve w
7: **if** $I_w \geq \sigma$ **then**
8: Insert w to S
9: $TYCOS_{\mathbf{LN}}(X'_T, Y'_T)$ ▹ Restart TYCOS$_{\mathbf{LN}}$
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 relationships are unknown. To overcome this challenge, we propose a robust method to set the correlation threshold based on the *normalized MI*:

$$0 \leq \tilde{I}_w = \frac{I_w}{H_w} \leq 1 \tag{18}$$

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, $\tilde{I}_w$ represents the fraction of the window's uncertainty reduced by the shared information I_w. The larger $\tilde{I}_w$, the more information is shared between the window's variables, and thus the stronger correlation. The normalized MI $\tilde{I}_w$ is always scaled between [0, 1], and thus provides an easier way for users to set the threshold σ.

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 σ. In this top-K filtering approach, σ starts with the MI value of the initial window w_0. As the search proceeds, the top-K list is filled, and σ 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 σ. The element with the least MI value in the top-K list will be replaced by this new window, and σ 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_4^1 and w_7^2 in Fig. 5, TYCOS will repeatedly revisit w_i because w_4^1 and w_7^2 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_i can differ from the current window w_i by only a small data partition w_{δ_i}, where w_{δ_i} is either removed from or added to w_i. For instance, in Fig. 5, w_8^1 differs from w_i by removing a w_{δ_1} data partition from w_i, whereas w_4^1 differs from w_i by adding a w_{δ_1} 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 $p_i = (x_i, y_i)$ is a square bounding box $R_i = (l_i, r_i, b_i, t_i)$, where l_i, r_i, b_i, t_i are its left-, right-, bottom-, and top-most indices, respectively, and are computed as $l_i = x_i - d$, $r_i = x_i + d$, $b_i = y_i - d$, $t_i = y_i + d$ where $d = \max(d_x, d_y)$.

Definition 7.2 *(Influenced marginal region (IMR))* The *IMRs* of point p_i are the marginal regions located within the nearest distance d_i in each dimension.

Fig. 8 illustrates these concepts. The *influenced region* of p_0 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 iff 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 iff 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 iff o is within IMR_x (or IMR_y) 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 iff o is within IMR_x (or IMR_y) 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. Effectiveness measures the method qualitatively by assessing the quality of extracted windows, while efficiency measures the method quantitatively in terms of its performance and accuracy.

8.1 Baseline methods

Effectiveness 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.

Efficiency evaluation: TYCOS runtime is compared against the Brute Force and MatrixProfile (which uses different window lengths) methods. In addition, different variants of TYCOS, including LAHC-based TYCOS ($\text{TYCOS}_\mathbf{L}$), $\text{TYCOS}_\mathbf{L}$ with noise theory applied ($\text{TYCOS}_\mathbf{LN}$), $\text{TYCOS}_\mathbf{L}$ with the proposed efficient MI computation ($\text{TYCOS}_\mathbf{LM}$), and $\text{TYCOS}_\mathbf{L}$ with both noise theory and efficient MI computation ($\text{TYCOS}_\mathbf{LMN}$), are compared against each other to illustrate the effectiveness of the proposed noise theory and MI computation technique. We do not compare AMIC against TYCOS quantitatively, however, as AMIC does not consider time delay correlations, and thus, has a different search space. PCC and MASS are also not considered for efficiency evaluation because they lack mechanisms to automatically search for correlated windows.

8.2 Parameter setting for TYCOS

TYCOS requires setting 5 parameters: correlation threshold σ, noise threshold ε, minimum window size $s_{\min}$, maximum window size $s_{\max}$, and maximum time delay $td_{\max}$. Among these, σ, $s_{\min}$, $s_{\max}$, and $td_{\max}$ are user parameters, while ε is a hyper parameter.

The value of σ determines the strength of extracted correlations. The larger the σ, the stronger the correlations. In our experiments, we set the value of σ 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 $td_{\max}$ 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 $td_{\max}$ to *24 hours* by assuming that weather events have impacts on other events only within *a day* duration. Table 2 lists the values of σ, $s_{\min}$, $s_{\max}$ and $td_{\max}$ we use in each dataset.

For the hyper parameter ε, we set $\varepsilon = \frac{1}{4}\sigma$ 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 $\varepsilon/\sigma = 0.25$ is chosen based on empirical studies we conduct

Table 1: **Identifying different types of correlation relations ($N(\mu, \sigma)$: normal distribution, $u \sim U(0, 1)$: uniform distribution)**

		td = 0 (No time delay)					td = 150 (With time delay)				
Relation	$y = f(x)$	PCC	MASS	MatrixProfile	AMIC	TYCOS	PCC	MASS	MatrixProfile	AMIC	TYCOS
Independent	$y \sim N(0, 1), x \sim N(3, 5)$	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Linear	$y = 2x + u, x \in [0, 10]$	✓	✓	✓	✓	✓	×	×	✓	×	✓
Exp.	$y = 0.01^{x+u}, x \in [-10, 10]$	✓	✓	×	✓	✓	×	×	×	×	✓
Quad.	$y = x^2 + u, x \in [-4, 4]$	×	✓	×	✓	✓	×	×	×	×	✓
Circle	$y = \pm\sqrt{3^2 - x^2 + u}, x \in [-3, 3]$	×	×	×	✓	✓	×	×	×	×	✓
Sine	$y = 2 * sin(x) + u, x \in [0, 10]$	×	×	×	✓	✓	×	×	×	×	✓
Cross	$y_1 = x + u, y_2 = -x + u, x \in [-5, 5]$	×	×	×	✓	✓	×	×	×	×	✓
Quartic	$y = x^4 - 4x^3 + 4x^2 + x + u, x \in [-1, 3]$	×	✓	×	✓	✓	×	×	×	×	✓
Square root	$y = \sqrt{x}, x \in [0, 25]$	×	✓	×	✓	✓	×	×	×	×	✓

Table 2: Parameters setting

Parameter	Energy datasets	Smart city datasets
Correlation threshold σ	0.3	0.2
Minimum window size $s_{\min}$	3 samples ≃ 3 mins	3 samples ≃ 15 mins
Maximum window size $s_{\max}$	10080 samples ≃ 7 days	4032 samples ≃ 14 days
Maximum time delay $td_{\max}$	2880 samples ≃ 2 days	288 samples ≃ 1 day

on different datasets, which consistently show that $\varepsilon/\sigma \simeq 0.25$ 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 σ, $s_{\max}$ and $td_{\max}$ on the performance of TYCOS.

8.3 Effectiveness 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, 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 ✓ sign denotes an identified relation, and the × sign denotes an 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 ≠ 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 × sign denotes no windows can be extracted.

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

Correlations	TYCOS	AMIC
(C1) *Kitchen* vs. *Dish Washer*	80, [0-4h]	25, 0h
(C2) *Kitchen* vs. *Microwave*	21, [0-1h]	5, 0h
(C3) *Clothes Washer* vs. *Dryer*	39, [10-30m]	×
(C4) *Bathroom Light* vs. *Kitchen Light*	14, [1-5m]	×
(C5) *Kitchen Light* vs. *Microwave*	11, [0-2m]	4, 0m
(C6) *Children Room Light* vs. *Living Room Light*	8, [15-40m]	×
(C7) *Precipitation* vs. *Collisions*	28, [0.5-2h]	×
(C8) *Wind Speed* vs. *Collisions*	23, [0.25-1h]	×
(C9) *Precipitation* vs. *Pedestrian Injured*	16, [0.5-2h]	×
(C10) *Wind Speed* vs. *Motorist Killed*	12, [0.25-1h]	×

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

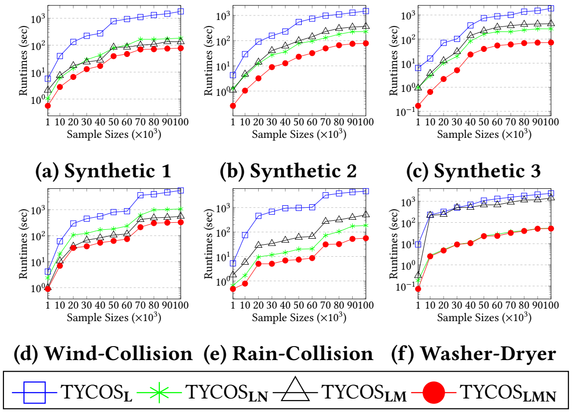


Figure 9: Runtime evaluation of TYCOS

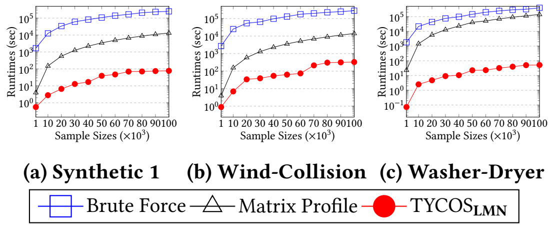


Figure 10: Brute Force, Matrix Profile, and TYCOS$_{\mathbf{LMN}}$

than on motorists or cyclists, while contrarily, wind has more impact on motorists and cyclists than pedestrians (C9, C10).

8.4 Efficiency 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: TYCOS$_{\mathbf{L}}$, TYCOS$_{\mathbf{LN}}$, TYCOS$_{\mathbf{LM}}$, TYCOS$_{\mathbf{LMN}}$, 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 TYCOS$_{\mathbf{LMN}}$ achieves the best performance among all versions. Its speedup w.r.t. TYCOS$_{\mathbf{L}}$ 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, TYCOS$_{\mathbf{LMN}}$, 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 TYCOS$_{\mathbf{LMN}}$ 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, TYCOS$_{\mathbf{L}}$ and TYCOS$_{\mathbf{LN}}$. Note that the efficient MI computation technique does not change the accuracy of TYCOS$_{\mathbf{L}}$, thus, TYCOS$_{\mathbf{LM}}$ and TYCOS$_{\mathbf{LMN}}$ 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

Table 4: Accuracy evaluation

	TYCOS$_{\mathbf{L}}$ vs. Brute Force		TYCOS$_{\mathbf{LN}}$ vs. TYCOS$_{\mathbf{L}}$	
Data Size	Synthetic Data	Real Data	Synthetic Data	Real Data
1K	96.2	95	100	100
10K	97.52	95.1	97.91	95.05
20K	94.08	91.7	98.19	97.78
30K	92.4	89.5	96.4	95.19
40K	97.85	95.1	98.17	97.01
50K	93.69	94.7	96.12	93.91
60K	95.49	94.8	97.1	97.78
70K	90.6	94.3	94.5	95.15
80K	88.75	91.02	96.21	95.8
90K	92.8	89.3	93.01	94.7
100K	93.1	94.7	95.8	94.94

Brute Force and TYCOS$_{\mathbf{L}}$ evaluates how accurate the LAHC approach on the TYCOS problem is, while the comparison between TYCOS$_{\mathbf{L}}$ and TYCOS$_{\mathbf{LN}}$ 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 this experiments.

Table 4 shows the average accuracy of TYCOS$_{\mathbf{L}}$ w.r.t. Brute Force, and of TYCOS$_{\mathbf{LN}}$ w.r.t. TYCOS$_{\mathbf{L}}$. Depending on the data sizes, TYCOS$_{\mathbf{L}}$ extracts from 88% to 98% similar windows compared to Brute Force, while TYCOS$_{\mathbf{LN}}$ extracts windows that are from 90% to 100% similar to TYCOS$_{\mathbf{L}}$.

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: ε, σ, $s_{\max}$, and $td_{\max}$, 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 ε**:** First, we examine how different values of ε affect the accuracy and runtime, using both synthetic and real-world data in Fig. 11. We can see, as the ratio ε/σ 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 ε/σ 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 ε. In Fig. 12, the accuracy and the runtime gain of each tested dataset are plotted together, with the ratio ε/σ on the x-axis. On the two tested datasets, i.e., energy and smart city datasets, we found that, when $\varepsilon/\sigma \in [0.05, 0.3]$, TYCOS$_{\mathbf{LN}}$ maintains an error rate less than 5%, while reducing the runtime up to 50%, compared to TYCOS$_{\mathbf{L}}$. Thus, our experimental setting $\varepsilon = \frac{1}{4}\sigma$ proved to be effective and robust. This threshold can be adjusted according to user's preference for accuracy.

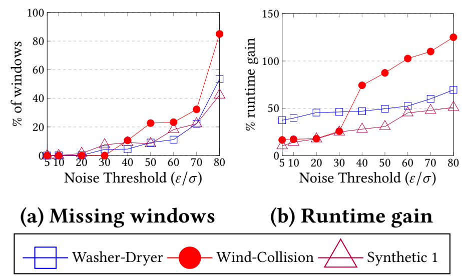
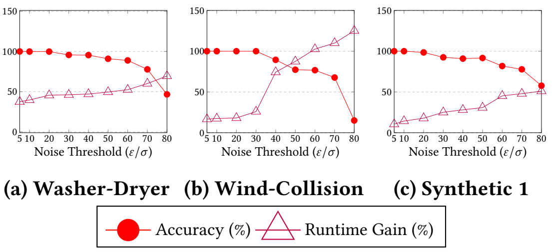
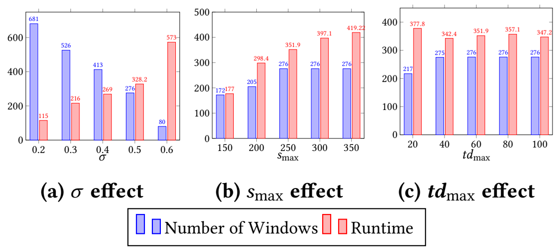


Figure 11: Effect of noise threshold ε



(a) Washer-Dryer (b) Wind-Collision (c) Synthetic 1

Figure 12: Trade-off analysis



(a) σ effect (b) $s_{\max}$ effect (c) $td_{\max}$ effect

Figure 13: Effect of σ, $s_{\max}$ and $td_{\max}$

B) Correlation threshold σ: We vary the values of σ to examine its effect, shown in Fig. 13a. We observe that, the correlations are stronger as σ 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 σ increases from 0.2 to 0.6, while the runtime increases from 115 to 573 seconds.

C) Window size $s_{\max}$ and time delay $td_{\max}$: We examine how $s_{\max}$ and $td_{\max}$ affect TYCOS. We found that, although $s_{\max}$ and $td_{\max}$ 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 $td_{\max}$, but an increasing runtime for $s_{\max}$. Fig. 13b and Fig. 13c illustrate this evaluation. Here, using the *(Snow, Collision)* datasets, TYCOS converges at $s_{\max} = 250$ and $td_{\max} = 60$, with 276 windows extracted when the convergence occurs. After the convergence, the runtime continues increasing as $s_{\max}$ goes beyond the value 250, while keeping similar values as $td_{\max}$ goes more than 60.

9 CONCLUSION AND FUTURE WORK

To our knowledge, TYCOS is the first comprehensive solution for the multi-scale time delay correlations search problem. 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: (1) integration of TYCOS and LAHC for multi-scale time delay correlations search, (2) the novel MI-based theory for noise identification, (3) the efficient MI computation technique to reduce computational redundancy. We perform an extensive evaluation on the effectiveness and efficiency of TYCOS, using both synthetic and real-world datasets. The evaluation shows that TYCOS can detect various types of relations in synthetic data, and find significant and interesting correlations in real-world data. 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 perform mining or infer causal effects from the extracted correlations.

ACKNOWLEDGMENT

This work has been partially supported by the DICYPS 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 Postdoc Program.

REFERENCES

[1] *Net-Zero Energy Residential Test Facility.* https://pages.nist.gov/netzero/data.html.
[2] *NYC Open Data.* https://opendata.cityofnewyork.us.
[3] A. Agresti and B. Finlay. 2014. *Statistical Methods for the Social Sciences.* Pearson Education Limited.
[4] A. Alawini, D. Maier, K. Tufte, and B. Howe. 2014. Helping scientists reconnect their datasets. In *SSDBM.*
[5] J.L. Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (1975).
[6] E.K. Burke and Y. Bykov. 2017. The late acceptance hill-climbing heuristic. *European Journal of Operational Research* 258, 1 (2017).
[7] F. Chirigati, H. Doraiswamy, T. Damoulas, and J. Freire. 2016. Data polygamy: the many-many relationships among urban spatio-temporal data sets. In *SIGMOD.*
[8] R. Cole, D. Shasha, and X. Zhao. 2005. Fast window correlations over uncooperative time series. In *ACM SIGKDD Proc.*
[9] T.M. Cover and J.A. Thomas. 2012. *Elements of information theory.* John Wiley & Sons.
[10] A. Das Sarma, L. Fang, N. Gupta, A. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu. 2012. Finding related tables. In *SIGMOD.*
[11] S. de Siqueira Santos, D.Y. Takahashi, A. Nakata, and A. Fujita. 2013. A comparative study of statistical methods used to identify dependencies between gene expression signals. *Briefings in bioinformatics* 15, 6 (2013).
[12] J.H Friedman, J.L. Bentley, and R.A. Finkel. 1976. An algorithm for finding best matches in logarithmic time. *ACM Trans. Math. Software* 3, SLAC-PUB-1549-REV. 2 (1976).
[13] M. Gribaudo, T.T.N. Ho, B. Pernici, and G. Serazzi. 2014. Analysis of the influence of application deployment on energy consumption. In *E2DC.*
[14] N. Ho, T.B. Pedersen, M. Vu, V.L. Ho, and C.A.N. Biscio. 2019. Efficient Bottom-Up Discovery of Multi-Scale Time Series Correlations Using Mutual Information. In *ICDE.*
[15] N. Ho and B. Pernici. 2015. A data-value-driven adaptation framework for energy efficiency for data intensive applications in clouds. In *IEEE SusTech.*
[16] N. Ho, H. Vo, and M. Vu. 2016. An adaptive information-theoretic approach for identifying temporal correlations in big data sets. In *IEEE BigData Proc.*
[17] N. Ho, H. Vo, M. Vu, and T.B. Pedersen. 2019. AMIC: An Adaptive Information Theoretic Method to Identify Multi-Scale Temporal Correlations in Big Time Series Data. *IEEE Trans. Big Data* (2019), 1–18.
[18] T.T.N. Ho. 2013. Activity recognition using smartphone based sensors. In *Master thesis.* Politecnico di Milano, Italy. http://hdl.handle.net/10589/85064.
[19] T.T.N. Ho. 2017. Towards sustainable solutions for applications in cloud computing and big data. In *Doctoral thesis.* Politecnico di Milano, Italy. http://hdl.handle.net/10589/131740.
[20] A. Kraskov, H. Stögbauer, and P. Grassberger. 2004. Estimating mutual information. *Physical review E* 69, 6 (2004), 066138.
[21] M. Middelfart, T.B. Pedersen, and J. Krogsgaard. 2013. Efficient Sentinel Mining Using Bitmaps on Modern Processors. *IEEE TKDE* 25, 10 (2013).
[22] A. Papana and D. Kugiumtzis. 2009. Evaluation of mutual information estimators for time series. *International Journal of Bifurcation and Chaos* 19, 12 (2009).
[23] K Pearson. 1895. Notes on Regression and Inheritance in the Case of Two Parents. (1895).
[24] R. Pochampally, A. Das Sarma, X.L. Dong, A. Meliou, and D. Srivastava. 2014. Fusing data with correlations. In *SIGMOD.*
[25] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. 2012. Searching and mining trillions of time series subsequences under dynamic time warping. In *ACM SIGKDD Proc.*
[26] S. Roy and D. Suciu. 2014. A formal approach to finding explanations for database queries. In *SIGMOD.*
[27] S.J Russell and P. Norvig. 2016. *Artificial intelligence: a modern approach.* Malaysia; Pearson Education Limited,.
[28] S. Salvador and P. Chan. 2007. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis* 11, 5 (2007).
[29] S. Shakil, J.C Billings, S.D Keilholz, and C.-H. Lee. 2018. Parametric dependencies of sliding window correlation. *IEEE TBE* 65, 2 (2018).
[30] M. Vejmelka and K. Hlaváčková-Schindler. 2007. Mutual information estimation in higher dimensions: A speed-up of a k-nearest neighbor based estimator. In *ICANNGA Proc.*
[31] C.-C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H.A. Dau, D.F. Silva, A. Mueen, and E. Keogh. Matrix profile I: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In *2016 ICDM.*
[32] P. Zhang, Y. Huang, S. Shekhar, and V. Kumar. 2003. Correlation analysis of spatial time series datasets: A filter-and-refine approach. In *PAKDD Proc.*