A Unified Approach for Multi-Scale Synchronous Correlation Search in Big Time Series

Nguyen Ho\textsuperscript{\textdagger}, Van Long Ho\textsuperscript{\textdagger}, Torben Bach Pedersen\textsuperscript{\textdagger}, Mai Vu\textsuperscript{\textdagger}, Christophe A.N. Biscio\textsuperscript{\textdagger}

\textsuperscript{\textdagger}Department of Computer Science, Aalborg University, Denmark
\textsuperscript{\textdagger}Department of Electrical & Computer Engineering, Tufts University, Medford, MA, USA
\textsuperscript{\textdagger}Department of Mathematical Science, Aalborg University, Denmark

Email: ntth@cs.aau.dk, vlh@cs.aau.dk, tbp@cs.aau.dk, mai.vu@tufts.edu, christophe@math.aau.dk

Abstract—The wide deployment of IoT sensors has enabled the collection of very big time series across different domains, from which advanced analytics can be performed to find unknown relationships, most importantly the correlations between them. However, current approaches for correlation search on time series are limited to only a single temporal scale and simple types of relations, and cannot handle noise effectively. This paper presents the integrated SYnchronous COrrelation Search (iSYCOS) framework to find multi-scale correlations in big time series. Specifically, iSYCOS integrates top-down and bottom-up approaches into a single auto-configured framework capable of efficiently extracting complex window-based correlations from big time series using mutual information (MI). Moreover, iSYCOS includes a novel MI-based theory to identify noise in the data, and is used to perform pruning to improve iSYCOS performance. Besides, we design a distributed version of iSYCOS that can scale out in a Spark cluster to handle big time series. Our extensive experimental evaluation on synthetic and real-world datasets shows that iSYCOS can auto-configure on a given dataset to find complex multi-scale correlations. The pruning and optimizations can improve iSYCOS performance up to an order of magnitude, and the distributed iSYCOS can scale out linearly on a computing cluster.

Index Terms—temporal correlation, hill climbing, sliding window, mutual information, top-down, bottom-up

1 INTRODUCTION

The massive spread of cheap IoT sensors has enabled the collection of big time series at unprecedented volume and velocity from different domains such as weather, energy, and transportation. Mining such big time series can help extract insights and unknown relationships from cross-domain datasets. One of the most fundamental analysis is to find correlations between time series, as the knowledge of how data are correlated with each other can enable more advanced analyses to support evidence-based decision making. For example, finding correlations between weather and transportation data can show how weather events like storm and rain affect traffic speed and use of taxis, from which public transport can be better scheduled; or knowing correlations between weather and energy production time series can help improve renewable energy forecasts. Furthermore, as correlation is one of the three building blocks to establish causal relations, finding correlations can thus help construct causal inference models.

Searching for correlations in big time series, however, is challenging. Specifically, an effective correlation search approach needs to deal with (1) multiple temporal scales, (2) complex relations, and (3) noise. First, correlations often appear at multiple temporal scales, for instance, correlations involved weather data might range from hours (e.g., during rain showers), to days or weeks (e.g., during a storm), depending on the type of weather events. Second, real-world time series exhibit different types of complex relations, including linear and non-linear, monotonic and non-monotonic, positive and negative, functional and non-functional. For example, stock prices or weather information exhibit non-linear relations, which rule out the use of traditional linear correlation metrics such as Pearson coefficients. Lastly, big time series from IoT sensors are inherently noisy due to errors introduced by sensors malfunction, transmission, collection and processing. Although correlation search has been studied extensively in the literature, current state-of-the-art approaches are limited either to a single temporal scale, simple types of relations, e.g., only linear ones, and/or in their inability to handle noise (see Section 2 for details). Moreover, it is a common misconception that correlations and similarities in time series are the same concept, but they are indeed different since correlations capture the dependencies between data distributions, while time series similarities capture the similar values of data points identified by distance measure such as Euclidean distance. For example, if values in one time series go up, while they go down in another, the two time series are not similar, instead they have a negative correlation. Therefore, methods used for time series similarities search such as Dynamic Time Warping are not suitable to identify correlations.

Contributions. The present paper addresses all three challenges above by presenting the unified framework iSYCOS (integrated SYnchronous COrrelation Search) that supports: (1) multiple temporal scales, (2) complex relations, and (3) handling noisy data. Specifically, we make the following contributions. First, iSYCOS searches for correlations by relying on the mutual information (MI) between time series windows. The metric of mutual information is robust to noise and underlying data distributions, and is able to capture complex correlation relations, including
linear and non-linear, monotonic and non-monotonic, positive and negative, functional and non-functional. Mutual information can also be computed for windows of any size, making it adaptive to multiple temporal scales. Second, by combining mutual information as the metric and a window-based approach, iSYCOS can automatically discover correlations at multiple temporal scales without requiring users to specify the time scales or window sizes. Window sizes are automatically adjusted based on mutual information value which is used as an indicator for a potential correlation. Third, we propose two window-based search methods, a top-down and a bottom-up one, to find correlations in time series datasets. The top-down approach, SYCOSTD, starts with the largest window size and gradually narrows it down, and is thus suitable for time series containing correlations across large time scales. Reversely, the bottom-up approach, SYCOSBU, starts with the smallest window size and gradually increases it, and is thus suitable for datasets where correlations are found at small time scales. Fourth, we propose a unified approach, iSYCOS, which integrates SYCOSTD and SYCOSBU, and automatically selects the most appropriate method for a given dataset. Fifth, based on probability theory, we propose a novel theory to identify and disregard noise when computing mutual information, hence enabling efficient pruning of non-interesting time intervals from the search and significantly improving the search speed. This also improves the robustness of iSYCOS in processing noisy data. Sixth, iSYCOS proposes effective optimizations to reuse mutual information computations across neighboring windows, thus improving MI computation speed and making iSYCOS fast and memory efficient. Seventh, we propose a distributed version of iSYCOS based on Apache Spark which is scalable to large data volumes. Eighth, we perform time and space complexity analyses for both SYCOSTD and SYCOSBU. And finally, we perform a comprehensive experimental evaluation using synthetic and real-world datasets from the energy and smart city domains, showing that iSYCOS can automatically select the best approach (top-down or bottom-up), is able to find complex correlations at multiple time scales, that the noise pruning and optimizations are effective and have high accuracy, and that iSYCOS can scale out linearly on a cluster.

Paper Outline. The paper is organized as follows. Section 2 discusses the related work. Section 3 formulates the window-based synchronous correlation search problem. Section 4 presents the background of mutual information and MI-based correlation. Section 5 presents SYCOSTD, SYCOSBU, and the integrated iSYCOS. Sections 6 and 7 discuss the noise pruning, the optimized MI computation, and the distributed iSYCOS. Section 8 presents the experimental evaluation and finally, Section 9 concludes and points to future work.

2 Related Work

Detecting correlations in time series has been extensively studied in the literature. Most of the proposed methods use traditional statistical metrics such as covariance and correlation coefficients (e.g., Pearson, Spearman) to measure correlations [6, 10, 17, 18, 20, 26, 27, 29, 29, 32, 33]. However, these metrics can only identify linear and monotonic relations. To capture more complex dependencies such as non-linear and non-monotonic, recent work [5, 11, 14, 16, 35, 42, 44, 51, 52] attempts to address the problem from a higher level. For example, in [14], Sarma et al. use relatedness to capture different kinds of related-relations, e.g., entity/schema complement, between data tables. Pochampally et al. [42] use joint precision and joint recall to measure the similarity between datasets. Alawini et al. [5] rely on the history and schema of datasets to map and link them. Roy et al. [44] use the concept of intervention to capture query relations. Sousa et al. [16] use the intrinsic dimension to detect correlated attributes in databases. Cai et al. in [8] use cohort analysis to look for causal explanations in the data. Yang et al. [32] use a residue metric that measures the difference between the actual and expected value of an object to capture object correlations in large datasets. Middelfart et al. [38] propose a bitmap-based approach to discover schema-level relationships in a multidimensional data cube. Chirigati et al. [11] propose a topology-based framework to identify relationships between spatio-temporal datasets during both regular and extreme events. Other work such as [51] uses Discrete Fourier Transform to detect local correlations in streaming data, with the main focus on the linear correlations.

Concerning correlation discovery techniques, the window-based technique has been used in [12, 23, 30, 31]. However, they adopt different correlation metrics, e.g., Schulz et al. [37] use the Spearman coefficient, while Cole et al. [12] use sketches. Other work such as [9] focuses on techniques for indexing and representing the time series, rather than the correlation search method itself.

Different from all the above work, our proposed iSYCOS framework uses MI as the correlation measure, and thus can discover not only linear and monotonic relations, but also non-linear and non-monotonic ones. By relying on a window-based technique to search for correlations, iSYCOS can find the exact time when the correlations occur, and thus provide richer information about the dependencies. Furthermore, we propose an MI-based theory that can identify noise in the data and thus, prune irrelevant data partitions during the search to improve its performance. We also optimize the MI computation to reuse information across windows, further speeding up the search.

The MI measure has been broadly used in numerous domains to achieve different goals, e.g., feature selection [18], clustering and mining [33], image alignment and registration [41], and network inference [57]. However, using MI to find correlations in a Big Data context is still a new area. Su et al. [48] use MI to analyze relationships of massive scientific datasets. However, they do not consider window-based correlations, but instead only focus on the overall correlations of variables. Keller et al. [34] propose the MISE algorithm to estimate MI for streaming data, but did not use this MI estimation to infer correlation.

In the present paper, the proposed iSYCOS framework is built upon the earlier versions of this work which investigated the two underlying approaches, bottom-up [21, 22] and top-down [24, 25]. In [21] and [22], we investigated how the Late Acceptance Hill Climbing algorithm [21] can be used to search for window-based correlations in a bottom-up fashion. In [24, 25], we proposed the top-down approach that uses a hierarchical multi-layer search
to discover correlated windows. In the present paper, we extend the previous work in the following ways. First, we unify the two approaches, top-down SYCOS$_{BD}$ and bottom-up SYCOS$_{BU}$, into one single framework iSYCOS. Second, we propose a set of efficiency scores to assess the performance of SYCOS$_{BD}$ and SYCOS$_{BU}$, and a selection algorithm using the efficiency scores to automatically analyze the input time series and select the most suitable method among SYCOS$_{BU}$ and SYCOS$_{BD}$ for a given input. Third, we integrate the novel MI-based noise theory, introduced previously in the bottom-up approach [21], into the top-down SYCOS$_{BD}$ to reduce the search space and improve its performance. Fourth, we develop a distributed version of iSYCOS using Apache Spark, allowing SYCOS$_{BU}$ and SYCOS$_{BD}$ to scale to big datasets on a cluster. Fifth, we conduct extensive experiments to compare their performance, showing which method is best for which specific scenarios.

3 PROBLEM FORMULATION

We now define the window-based correlations, and formulate the Synchronous Correlation Search (SYCOS) problem.

Definition 3.1 (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 chronologically ordered.

The observation time period $T = [t_1, t_n]$ contains $n$ time steps $t_i$, each recording a data value $x_i \in X_T$, where $t_1$ and $t_n$ denote the first and the last time step of $T$, respectively. We say $X_T$ has length $n$ if $X_T$ contains $n$ data values.

Definition 3.2 (Time window) A time window $w = [s, e]$ is a temporal sub-interval of $T$ that records the events of $X_T$ from the start time $s$ to the end time $e$, and forms a sub time series $X_w = \{x_s, x_{s+1}, ..., x_e\} \subset X_T$. We say that $w$ has size $m$, denoted as $|w| = m$, if $X_w$ contains $m$ data values.

Definition 3.3 (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 time interval $T$. A tuple $(x_i, y_i) \in (X_T, Y_T)$ records the data values of $X_T$ and $Y_T$ at the same time instance $t_i$.

Definition 3.4 (Pair of time windows) A pair of time windows $(w_{X}, w_{Y}) = ([t_{x_1}, t_{x_2}], [t_{y_1}, t_{y_2}])$ is a pair of temporal sub-intervals of $T$ that records the events of $X_T$ and $Y_T$ during $[t_{x_1}, t_{x_2}]$ and $[t_{y_1}, t_{y_2}]$, respectively. The pair $(w_{X}, w_{Y})$ is synchronous if $t_{x_1} = t_{y_1}$ and $t_{x_2} = t_{y_2}$. For simplicity, the synchronous pair $(w_{X}, w_{Y})$ is called a time window of $(X_T, Y_T)$, and denoted as $w_{X,Y} = [t_{s}, t_{e}]$ where $t_s$ and $t_e$ are the start and end time of the pair, and $t_s = t_{x_1} = t_{y_1}$ and $t_e = t_{x_2} = t_{y_2}$.

Definition 3.5 (Correlated time window) Consider a pair of time series $(X_T, Y_T)$ measured during the time interval $T$, and a time window $w_{X,Y}$ of $(X_T, Y_T)$. Let $f(.)$ be a predefined correlation function, and $\sigma$ be a predefined correlation threshold. The two time series $X_T$ and $Y_T$ are said to be correlated within the window $w_{X,Y}$ if $X_w$ and $Y_w$ are correlated according to $f(.)$, i.e., $f(X_w, Y_w) \geq \sigma$.

The function $f(.)$ can be any function that measures data dependencies, for example, Pearson correlation coefficient, Spearman correlation, Euclidean distance, or Mutual Information. Based on the above definitions, we formulate the synchronous correlation search problem as follows.

Synchronous Correlation Search (SYCOS): Let $(X_T, Y_T)$ be a pair of time series measured during the time interval $T$, $s_{\text{min}}$ and $s_{\text{max}}$ be the minimum and maximum sizes that a time window $w_{X,Y}$ can have, $f(.)$ be a correlation function, and $\sigma$ be the predefined correlation threshold. Then SYCOS finds a set $S$ of all non-overlapping time windows $w_{X,Y} = [t_s, t_e]$ such that $t_1 \leq t_s < t_e \leq t_n$ and $s_{\text{min}} \leq |w_{X,Y}| \leq s_{\text{max}} \land f(X_w, Y_w) \geq \sigma \land \forall w_i, w_j \in S: w_i \cap w_j = \emptyset$.

The SYCOS problem aims to find all non-overlapping time windows of a given time series pair such that their sub time series are correlated according to the correlation function and the predefined threshold.

4 MUTUAL INFORMATION-BASED CORRELATION

In Section 3, we use a general correlation function $f(.)$ to define correlated time windows. In this section, we discuss our choice of using mutual information as a correlation measure, and argue why it is a good fit for the correlation search problem.

4.1 Mutual Information

Mutual information 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 (pmfs) $p(x)$, $p(y)$, and the joint distribution $p(x, y)$, the mutual information between $X$ and $Y$ is defined as

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

(1)

Intuitively, $I(X; Y)$ represents the reduction of uncertainty of one variable (e.g., $X$) given the knowledge of another variable (e.g., $Y$) [13]. 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. Mutual information possesses multiple appealing properties that lead to its widespread use in many different fields, including machine learning, neuroscience, and data compression [27]. Given the mutual information $I(X; Y)$, the following properties hold: (1) Non-negativity: $I(X; Y) \geq 0$; (2) Invariance under one-to-one transformations: $I(X; Y) = I(U(X); V(Y))$ where $U$ and $V$ are homeomorphisms that create homeomorphic mappings on $X$ and $Y$; (3) Chain rule expansion: $I(X_1, ..., X_n; Y) = \sum_{i=1}^{n} I(X_i; Y | X_{i-1}, ..., X_1)$.

The MI value is equal to zero if and only if the considered variables are statistically independent, and otherwise always positive if there exists any kind of dependency (e.g., functional and non-functional, linear and non-linear) [15]. This property makes MI a versatile measure to capture correlations in noisy data sets which often exhibit a high degree of bias and abnormality, causing their relationships to often be arbitrary and non-linear.

4.2 Estimating Mutual Information

Although MI is an important measure of information for many applications, applying MI in practice is challenging
due to the difficulty in estimating probability distributions. As finding a more efficient method to estimate MI from collected samples is still an active research problem, in this work, we adopt the MI estimator proposed by Kraskov et al. [36] (hereafter called KSG) for several reasons: (1) KSG outperforms other estimators (e.g., histogram, kernel density estimation) in terms of computational efficiency and accuracy [39]. (2) KSG uses $k$-nearest neighbor approximation and thus is data efficient (i.e., it does not require very large sets of samples), is adaptive and has minimal bias [36].

The main idea of KSG is that 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 [36]. Specifically, KSG computes the probability distribution for the distance between each data point and its $k$th nearest neighbor. For each data point, it first 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 within the distance $d_k$ is estimated by counting the number of data points that fall inside $d_k$. This leads to the computation of MI between $X$ and $Y$ as [35]:

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

(2)

where $\psi$ is the digamma function, $k$ is the number of nearest neighbors, $(n_x, n_y)$ is 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 $\psi$ 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. In practice, mutual information can be computed using empirical distributions as follows.

**Definition 4.1 (Empirical distribution)** Let $X_T$ be a time series measured during a time interval $T$. A distribution $\hat{p}$ is an empirical distribution of $X_T$, denoted as $\hat{p}(x_i)$, if $\hat{p}$ is estimated from the empirical measured values $x_i \in X_T$, $i = 1,...,n$.

**Definition 4.2 (Mutual information of a time window)** Let $(X_T, Y_T)$ be a pair of time series measured during the time interval $T$, and $w_{X,Y}$ be a time window of $(X_T, Y_T)$. The mutual information between $X$ and $Y$ within $w_{X,Y}$ can be estimated using the empirical distributions $\hat{p}$ of data within this window as

$$I_{w_{X,Y}} = I(X_w; Y_w) \approx \sum_{y_i \in Y_w} \sum_{x_i \in X_w} \hat{p}(x_i, y_i) \log \frac{\hat{p}(x_i, y_i)}{\hat{p}(x_i)\hat{p}(y_i)}$$

(3)

5. **The Unified SYCOS Search Framework**

We now present our unified framework iSYCOS for synchronous correlation search. We start by describing different scenarios where correlated time windows are distributed differently on a given pair of time series, and how different search approaches are beneficial for each of those scenarios.

Figure 1 outlines three scenarios of correlated time windows in the time series. The first scenario in Figure 1a illustrates a situation where correlated time windows are large and located close or next to each other. In such a scenario, a search strategy that starts with a large window, and only reduces the window size when needed, is more efficient to identify correlated windows. We call this search strategy a top-down approach. The second scenario in Figure 1b is when correlated windows are small and sparsely distributed. In this scenario, a search strategy that starts from a small window, and incrementally extends the window size only when needed, is faster to locate correlated windows. We call this search strategy a bottom-up approach. Finally, the third scenario in Figure 1c is when correlated windows are both small and large, and are neither close nor far away from each other. In this scenario, we need a strategy to select between bottom-up and top-down search.

The following sections will introduce the top-down and bottom-up approaches separately. We then present a selection algorithm that automatically evaluates and selects the most efficient search approach between top-down and bottom-up for a given time series pair.

5.1 **SYCOSTD: Top-Down Search**

The top-down approach SYCOSTD uses a breadth-first search strategy to search for correlated time windows: it starts with a large window, and reduces the size when needed as the search progresses. Intuitively, SYCOSTD assumes that correlations exist in large windows, and thus, it is more efficient to start with those.

5.1.1 **Multi-layer hierarchical search space**

To perform the top-down search, SYCOSTD divides the search space into multiple hierarchical layers, each corresponding to one specific window size. The temporal scale is used to keep track of window sizes among different layers.

**Definition 5.1 (Temporal scale)** A temporal scale represents the size of windows belonging to the same layer in a multi-layer hierarchical search space.

Examples of temporal scale are year, month, week, day, and so on. Users can define the temporal scale using their domain knowledge. For instance, windows with temporal scale year contain time series of one year, while windows of day temporal scale contain time series of one day. Fig. 2 depicts the multi-layer hierarchical search space. Here, the top layer $L_1$ has the coarsest temporal scale, corresponding to the largest windows. The bottom layer $L_n$ has the finest temporal scale, corresponding to the smallest windows.

5.1.2 **Top-down search with sliding window**

Given the multi-layer search space, initially, SYCOSTD starts out at the top layer $L_1$, using the largest window size $s_{\text{max}}$. In each layer, SYCOSTD uses a sliding window to slide over the time series, identifying correlated windows and filtering out uncorrelated ones. Specifically, to determine whether the time series in a time window $w_i$ are correlated, the MI $I_{w_i}$ is computed and compared against the threshold $\sigma$: $w_i$ is a correlated window if $I_{w_i} \geq \sigma$, and uncorrelated otherwise. The result set $S$ is used to collect correlated windows, while uncorrelated ones are passed down to the next lower layer of the search space. As the search progresses, SYCOSTD traverses to lower layers until it reaches the lowest one $L_n$ which has the smallest window size $s_{\text{min}}$.

In the same layer, two consecutive windows $w_i$ and $w_{i+1}$ can be either disjoint or overlapping depending on whether significant correlation exists in one of them. More specifically, if $w_i$ is a correlated window, then $w_i$ and $w_{i+1}$
are disjoint. In contrast, if \( w_i \) is an uncorrelated window, then \( w_{i+1} \) is created by shifting \( w_i \) by a predefined \( \delta \)-step, making \( w_i \) and \( w_{i+1} \) overlap each other. We illustrate this sliding window technique in Fig. 2.

SYCOS\(_{TD}\) starts out at \( L_1 \) with the first window \( w_0 = [t_{s_0}, t_{e_0}] \) of size \( s_{max} \). The MI value \( I_{w_0} \) is computed and compared against \( \sigma \) to determine whether \( w_0 \) is a correlated window. In this example, we assume \( I_{w_0} < \sigma \), and thus, \( w_0 \) is uncorrelated. SYCOS\(_{TD}\) passes \( w_0 \) down to layer \( L_2 \) and moves to the next window \( w_1 \in L_1 \). To define \( w_1 \), SYCOS\(_{TD}\) shifts \( w_0 \) by a \( \delta \)-step, forming \( w_1 = [t_{s_1}, t_{e_1}] \) as \( t_{s_1} = t_{s_0} + \delta \) \( \land \ t_{e_1} = t_{e_0} + \delta \). The MI of \( w_1 \) is then evaluated, which results in \( I_{w_1} < \sigma \). Here, \( w_0 \) and \( w_1 \) are two consecutive and uncorrelated windows, thus, their indices are merged into one single uncorrelated data partition \( [t_{s_0}, t_{e_1}] \). This uncorrelated data partition will be used by SYCOS\(_{TD}\) in the next search iteration in the lower layer.

Next, SYCOS\(_{TD}\) moves to \( w_2 \) which is formed by shifting \( w_1 \) a \( \delta \)-step. Assume that at \( w_2 \), we have \( I_{w_2} \geq \sigma \), hence, the time series in \( w_2 \) are correlated. The window \( w_2 \) is inserted into the result set \( S \), after which the uncorrelated data partition \( [t_{s_0}, t_{e_1}] \) previously formed by \( w_0 \) and \( w_1 \) is updated to \( [t_{s_2}, t_{e_2}] \), indicating that only the time series from \( t_{s_2} \) to \( t_{e_2} \) are uncorrelated. The search moves on to the next window \( w_3 = [t_{s_3}, t_{e_3}] \). Here, since \( w_2 \) is a correlated window, \( w_3 \) starts right after \( w_2 \) with \( t_{s_3} = t_{e_2} + 1 \). The procedure is repeated until the entire time series in \( L_1 \) is scanned.

At \( L_2 \) and lower layers, SYCOS\(_{TD}\) repeats the search procedure on uncorrelated data partitions passed down from the above layers, using progressively smaller windows until the lowest layer \( L_n \) is scanned.

Algorithm 1 gives the pseudocode for SYCOS\(_{TD}\). In line 1, SYCOS\(_{TD}\) starts at \( L_1 \) with windows of size \( s_{max} \). Lines 6-14 perform sliding window search on each layer. Line 17 reduces the temporal scale in order to move to the next lower layer. SYCOS\(_{TD}\) stops when it finishes the search of the lowest layer \( L_n \) (line 2).

5.2 SYCOS\(_{BU}\): Bottom-Up Search

In contrast to the top-down approach, the bottom-up approach SYCOS\(_{BU}\) starts with a small window, and incrementally enlarges the window size as the search progresses. SYCOS\(_{BU}\) assumes that correlations exist in small windows, and thus, it is more efficient to start with those.

5.2.1 Late Acceptance Hill Climbing

SYCOS\(_{BU}\) employs Late Acceptance Hill Climbing (LAHC) 7, an extension of the classic Hill Climbing (HC) [45], to navigate through the search space while looking for correlated windows. Particularly, given a target function \( f \) and a current solution \( S \) of \( f \), LAHC tries to improve \( S \) by exploring the neighborhood of \( S \) to find new solutions that are better than \( S \). If a better solution is found (according to the predefined objectives), the current solution \( S \) is replaced by this new solution \( S_{new} \), and the process is repeated until no further improvement can be made. Compared to classic HC, LAHC is different in terms of its acceptance rule, i.e., it can accept a solution \( S_{new} \) 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 this \( L_h \) array to justify the goodness of a candidate solution.

5.2.2 Applying LAHC to SYCOS\(_{BU}\)

We consider the problem of searching for synchronous correlations using LAHC as a maximization problem. Specifically, the target function of LAHC is a maximize function, and our goal is to search for the time windows where their MIs are locally maximal. To explain why local optima are in fact the solutions of SYCOS\(_{BU}\), consider Fig. 4 that plots the fluctuation of MI values for a given pair of variables. Here, the \( y \)-axis represents the MI values of the time windows on the \( x \)-axis. Given the correlation threshold \( \sigma \) (red line in Fig. 4), the three windows which correspond to the three maximal points (in red) indicate highly correlated areas (above \( \sigma \)). These windows can be found by first identifying the three peak points which represent the locally maximal MI values, and then extending the located points to the surrounding areas to determine the exact indices of the windows. Such windows are guaranteed to satisfy the correlation condition \( I_w \geq \sigma \). Since an LAHC-based method guarantees local optimal solutions, it is an ideal technique for SYCOS.

Definition 5.2 (\( \delta \)-neighbor) Let \( (X_T, Y_T) \) be a pair of time series of length \( n \) measured during the time interval \( T \), and \( w = [t_s, t_e] \) be a time window of \( (X_T, Y_T) \). A window
Fig. 2: SYCOS

$w' = [t'_s, t'_e]$ is a $\delta$-neighbor of $w$ if its start time $t'_s$ and/or its end time $t'_e$ differs by a $\delta$ value from $t_s$ and $t_e$, respectively. i.e., $t'_s = t_s + \delta \lor t'_e = t_e - \delta$, where $\delta$ is a pre-defined moving step such that $1 \leq \delta \leq n \land s_{\min} \leq |w| \leq s_{\max}$.

Definition 5.3 ($\delta$-neighborhood) Let $w = [t_s, t_e]$ be a time window of $(X_T, Y_T)$. A $\delta$-neighborhood $N$ of $w$, denoted as $N_{\delta}$, is the surrounding area of $w$ formed by all $\delta$-neighbors $w' = [t'_s, t'_e]$ of $w$.

We illustrate the neighborhood concept in Fig. 3. Here, each point in the grid represents a time window where the x-axis indicates the start time, and the y-axis indicates the end time. Consider the window $w$ in blue. The nearest $\delta$-neighborhood of $w$, called the 1-neighborhood $N_1$, is the area formed by the eight windows in light purple color surrounding $w$. Each window in this neighborhood differs from $w$ by one $\delta$ step, either by its start index, or its end index, or both. Going further from $w$, the next neighborhood of $w$, called the 2-neighborhood $N_2$, is the 16 windows in light blue color. Each $\delta$-neighborhood forms an area where LAHC will iteratively look for solution candidates of a given window $w$. We provide the pseudocode of SYCOS$_{BU}$ in Algorithm 2, and explain the details below.

Given a pair of time series $(X_T, Y_T)$, and the target function $I_w$ to be maximized, SYCOS$_{BU}$ starts with an initial solution $w = w_0$ where $|w_0| = s_{\min}$ (Algorithm 2 line 2), and evaluates the goodness of $w$ by computing the MI $I(w_0)$ (line 3). To maximize $I_w$, SYCOS$_{BU}$ first searches for a better solution in its nearest neighborhood $N_1$. It does so by computing $I(w')$ for each neighbor window $w' \in N_1$, and selects the best neighbor $w_{\text{best}}$ that has the highest MI in the neighborhood (lines 7-11). Then, SYCOS$_{BU}$ determines whether $w_{\text{best}}$ is better than the current solution $w$ using the following policies:

- **Policy P1**: If $I_{w_{\text{best}}} \geq I_w$ or $I_{w_{\text{best}}} \geq I_{w_{h}}$, where $w_{h} \in L_{h}$, then $w_{\text{best}}$ is better than $w$ and thus, $w$ is replaced by $w_{\text{best}}$ (lines 13-15).
- **Policy P2**: If $I_{w_{\text{best}}} < I_w$ and $I_{w_{\text{best}}} < I_{w_{h}}$, where $w_{h} \in L_{h}$, there is no better solution found in the considered neighborhood, thus, no improvement can be made (lines 16-18).

In policy P1, a better solution is found, thus, SYCOS$_{BU}$ moves to this new solution $w = w_{\text{best}}$, and repeats the exploration process on the new $w$. In policy P2, no better solution is found. In this case, SYCOS$_{BU}$ checks the stopping conditions and determines whether it should continue exploring a larger neighborhood or stopping the search immediately. Ideally, SYCOS$_{BU}$ will stop immediately when no better solution can be found in the current neighborhood. However, to avoid situations where a temporary setback stops the search too early, we use an idle period to measure the number of non-improvements observed. The search will stop when it reaches the pre-defined max idle period $T_{\text{max-idle}}$ (line 6). The value $I_w$ at the stopping point is the locally maximal value of the target function, and is compared against the correlation threshold $\sigma$. The window $w$ is inserted into the result set $S$ if $I_w \geq \sigma$ (line 24).

When SYCOS$_{BU}$ stops, the time series might not be scanned entirely. In that case, SYCOS$_{BU}$ restarts on the remaining part of the data, looking for new local optima, until the entire time series is scanned (line 25).

Initial solution: The first window $w_0$ of SYCOS$_{BU}$ can be at the beginning of the time series, or at an arbitrary position within the time series.

The history list $L_{h}$: SYCOS$_{BU}$ maintains a history list $L_{h}$ of the most recently accepted solutions and uses it to justify the goodness of a solution candidate. In our implementation, SYCOS$_{BU}$ follows the random policy when selecting and updating an item in $L_{h}$: to justify a solution candidate, SYCOS$_{BU}$ randomly picks an element from the list and compares to the candidate; similarly, a random element from $L_{h}$ is selected for updating. This random policy helps SYCOS$_{BU}$ escape plateaus in the search space by creating "randomness" while it moves.

5.3 iSYCOS: Integrating SYCOS$_{TD}$ and SYCOS$_{BU}$

In this section, we integrate SYCOS$_{TD}$ and SYCOS$_{BU}$ into a single integrated framework iSYCOS that automatically
Finally, iSYCOS computes the average runtimes on each data partition and considers two factors: the runtime and the size of extracted windows. More specifically, on each data partition into consideration two factors: the runtime and the size of small or large windows:

\[ w \in \left\{ \text{small or large windows} \right\} \]

Next, based on the window size, iSYCOS classifies windows into small or large windows:

\[ w \in \left\{ \text{small or large windows} \right\} \]

Next, we randomly select \( n_s \) windows and SYCOS on the extracted windows as

\[ \text{score}_{TD} = \alpha \times \frac{1}{\bar{t}_{TD}} + (1 - \alpha) \times n_s^{TD} \]

where \( \alpha \) and \((1 - \alpha)\) are the weights placed on the runtime, and the number of large windows of SYCOS_{TD} or the number of small windows of SYCOS_{BU}, respectively.

The top-down score computed in Eq. (4) is the weighted sum of the inverse of the average runtime, and the number of large windows. The higher the score_{TD}, the lower the average runtime and the more large windows extracted by SYCOS_{TD}. Similarly for the bottom-up score, the higher the score_{BU}, the lower the average runtime and the more small windows extracted by SYCOS_{BU}. By adjusting the value of \( \alpha \), users can put more weight on the factor that they prefer to be more important, i.e., the runtime or the extracted windows.

However, since the runtime and the number of extracted windows are measured by different scales, we normalize them between \([0, 1]\) so that they are weighted properly:

\[ \tilde{t}_{TD} = \frac{t_{TD}}{t_{BU} + t_{TD}}; \quad \tilde{t}_{BU} = \frac{t_{BU}}{t_{BU} + t_{TD}} \]

\[ \tilde{n}_s^{TD} = \frac{n_s^{TD}}{n_s^{BU} + n_s^{TD}}; \quad \tilde{n}_s^{BU} = \frac{n_s^{BU}}{n_s^{BU} + n_s^{TD}} \]

where \( \tilde{t}_{TD} \) and \( \tilde{t}_{BU} \) are the normalized average runtimes, and \( \tilde{n}_s^{TD} \) and \( \tilde{n}_s^{BU} \) are the normalized number of large and small windows of SYCOS_{TD} and SYCOS_{BU}, respectively.

Finally, we replace the normalized scores computed in Eqs. (6) and (7) into Eqs. (4) and (5), and obtain the normalized efficiency scores for SYCOS_{TD} and SYCOS_{BU} as

\[ \text{nscore}_{TD} = \alpha \times \tilde{t}_{TD} + (1 - \alpha) \times \tilde{n}_s^{TD} \]

\[ \text{nscore}_{BU} = \alpha \times \tilde{t}_{BU} + (1 - \alpha) \times \tilde{n}_s^{BU} \]

Using nscore_{TD} and nscore_{BU}, iSYCOS selects SYCOS_{TD} over SYCOS_{BU} if nscore_{TD} > nscore_{BU}, and selects SYCOS_{BU} over SYCOS_{TD} otherwise.

Algorithm 3 provides the outlines of integrated iSYCOS framework. First, it starts with random sampling to select \( m \) partitions for selection analysis (line 1). Lines 2-5 execute SYCOS_{TD} and SYCOS_{BU} on each selected data partition and record their runtimes and extracted windows. Lines 6-10 compute the efficiency scores which decide which search method is selected (lines 11-15).

5.3.3 Space complexity analysis

The time and space complexities of SYCOS_{TD} and SYCOS_{BU} depend on the number of feasible windows \( n_w \) in the search space, and the time \( t_w \) to compute the MI value of a window. Here, we analyze the space complexity of SYCOS_{TD} and SYCOS_{BU}, and save the discussion of time complexity for Section 7.2.

For SYCOS_{TD}, the total number of feasible windows in the search space is \( n_w = \sum_{i=m}^{\max}(N-i) \) where \( N \) is the length of input time series and \( m \leq N \). Thus, the space complexity of SYCOS_{TD} is \( O(N^2) \).
Algorithm 3 iSYCOS: Integrated SYCOS

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

Params: \(\sigma\): correlation threshold,
\(s_{\min}, s_{\max}\): minimum and maximum window sizes,
\(m, M\): number of selected and total sub-partitions,
\(\delta_{TD}\): sliding step of SYCOS\(_{TD}\),
\(\delta_{BU}\): moving step of SYCOS\(_{BU}\),
\(\alpha\): weight placed on runtime factor

Output: \(S\): set of non-overlapping windows with \(MI \geq \sigma\)

1: \(mList \leftarrow \text{randomSampling}(M)\) \(\triangleright \) select \(m\) partitions from \(M\)
2: for \(m \in mL\) do
3: \(t_{TD}\leftarrow W_{TD} \rightarrow \text{SYCOS}_{TD}(m)\) \(\triangleright \) Top-down search
4: \(t_{BU}\leftarrow W_{BU} \rightarrow \text{SYCOS}_{BU}(m)\) \(\triangleright \) Bottom-up search
5: end for
6: \(t_{TD}\leftarrow \text{averageRuntime}(t_{TD})\) \(\triangleright \) top-down average runtime
7: \(t_{BU}\leftarrow \text{averageRuntime}(t_{BU})\) \(\triangleright \) bottom-up average runtime
8: \(n_{TD}, n_{TD}^\top \leftarrow \text{countWindows}(W_{TD}), i = 1, \ldots, m\)
9: \(n_{BU}, n_{BU}^\top \leftarrow \text{countWindows}(W_{BU}), i = 1, \ldots, m\)
10: \(\text{nscore}_{TD}, \text{nscore}_{BU} \leftarrow \text{normalizedScore}(t_{TD}, n_{TD}^\top, n_{TD}, \alpha)\)
11: if \(\text{nscore}_{TD} > \text{nscore}_{BU}\) then
12: \(S \leftarrow \text{SYCOS}\(_{TD}(X_T, Y_T, \sigma, s_{\min}, s_{\max}, \delta_{TD})\) \(\triangleright \) select top-down
13: else
14: \(S \leftarrow \text{SYCOS}\(_{BU}(X_T, Y_T, \sigma, s_{\min}, s_{\max}, \delta_{BU})\) \(\triangleright \) select bottom-up
15: end if
16: return \(S\)

For SYCOS\(_{BU}\), the total number of feasible windows in the search space is \(n_w = (s_{\max} - s_{\min})N \sum_{i=1}^{T_{\text{max}}^\top} 8 \times i\) where \(N\) is the length of the input time series, and \(s_{\max} \leq N\). Since we have \(\sum_{i=1}^{T_{\text{max}}^\top} i \sim T_{\text{max}}^\top^2\) the space complexity of SYCOS\(_{BU}\) is \(O(N^2T_{\text{max}}^\top)\).

However, we note that the above complexity analyses is only for the worst case scenarios. In practice, due to the filtering of correlated windows during the search, the numbers of windows explored by SYCOS\(_{TD}\) and SYCOS\(_{BU}\) are much smaller.

5.4 Setting the Correlation Threshold

Since MI measures the dependencies between variables, its magnitude indicates the strength of correlations, i.e., the larger the MI value, the stronger the correlation. However, MI is an unbounded measure, i.e., \(0 \leq I_w < \infty\). Thus it is difficult to interpret the correlation strength and to properly set the correlation threshold. Thus, we now propose a robust method to set the threshold \(\sigma\) based on the normalized MI:

\[
0 \leq \tilde{I}_w = \frac{I_w}{H_w} \leq 1 \tag{10}
\]

where \(I_w\) is the MI, and \(H_w\) is the entropy of \(w\), defined as

\[
H_w = H(X; Y_w) = \sum_{y \in Y_w} \sum_{x \in X_w} \hat{p}(x, y) \log \frac{\hat{p}(x, y)}{\hat{p}(x)\hat{p}(y)} \tag{11}
\]

In Eq. (10), the window entropy \(H_w\) represents the amount of uncertainty intrinsically reduced by the shared information \(I_w\). Larger the \(I_w\), the more information is shared between the window’s variables, and thus the stronger the 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 \(\sigma\).

6 Noise-based Pruning in SYCOS

In this section, we propose a novel MI-based theory to identify noise in the input time series, and show how to incorporate this theory into SYCOS\(_{TD}\) and SYCOS\(_{BU}\) to filter noises and improve the search performance.

6.1 MI-based Theory to Identify Noises

When SYCOS\(_{TD}\) and SYCOS\(_{BU}\) search for correlated windows, they might visit the same data partition multiple times. This is because data partitions can be shared between neighboring windows in SYCOS\(_{BU}\), or between consecutive windows in SYCOS\(_{TD}\). For example, consider the SYCOS\(_{BU}\) search space in Fig. 3. Let \(w = [t_s, t_e]\) (blue point) be the current window, and \(N_1\) and \(N_2\) be its two nearest neighborhoods. The neighboring windows of \(w\) in \(N_1\) and \(N_2\) share data among each other. More specifically, \(w_{N_1} = [t_{s_{N1}}, t_{e_{N1}}] \in N_1\) is created from \(w = [t_s, t_e]\) by extending the end index \(t_e\) of \(w\) by a \(\delta_1\) step. Thus, \(w_{N_1}\) shares with \(w\) the data partition \([t_s, t_e]\) which is exactly the window \(w\) itself. Similarly, \(w_{N_2} = [t_{s_{N2}}, t_{e_{N2}}] \in N_2\) is created from \(w\) by enlarging the end index \(t_e\) of \(w\) by a \(\delta_2\) step, with \(\delta_2 > \delta_1\). Thus, \(w_{N_2}\) shares with \(w\) a partition \([t_s, t_e]\), while sharing with \(w_{N_1}\) a partition \([t_s, t_{e_{N2}}]\).

Similarly, consecutive windows in SYCOS\(_{TD}\) also share data among each other. In Fig. 2 when SYCOS\(_{TD}\) moves from \(w_0 = [t_{s_0}, t_{e_0}]\) to \(w_1 = [t_{s_1}, t_{e_1}]\) in layer \(L_1\), SYCOS\(_{TD}\) creates an overlapping data partition \([t_{s_1}, t_{e_1}]\) that is shared between the two windows.

The shared data partitions create redundant computation and degrade the search performance. To reduce this redundancy, it is important to identify shared data partitions which do not provide important information about correlations. Such data partitions are considered as noises, and can thus be removed from the search space without affecting the final search results. The presence of such data partitions is demonstrated in the following example.

Consider Fig. 5 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. 5 it is clear that by excluding the data partition \([0, 5]\) from the search, the MI values of subsequent windows increase and are generally larger than when including it. Thus, the data partition \([0, 5]\) provides no information about correlations between the times series, and can be considered as noise that should be eliminated from the search exploration. We exploit the MI properties to establish a noise identification principle below.

Definition 6.1 (Mixture distribution) Let \(X\) and \(U\) be discrete random variables with the corresponding p.m.f.s \(p_x(x)\) and \(p_u(u)\). 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 \(p_u(u)\) and is written as \(Z = X \circ \theta U\).

Theorem 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.f.s. Let \(Z = X \circ \theta U\) and \(W = Y \circ \theta V\) where \(\circ\) 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: \(Z = X \circ \theta U\) and \(W = Y \circ \theta V\). 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:

$$p_Z(x) = P(Z = X)p_X(x) = \theta p_X(x)$$  \hspace{1cm} (12)$$

$$p_Z(u) = P(Z = U)p_U(u) = (1 - \theta)p_U(u)$$ \hspace{1cm} (13)

Similarly, we have:

$$p_W(y) = P(W = Y)p_Y(y) = \eta p_Y(y)$$ \hspace{1cm} (14)$$

$$p_W(v) = P(W = V)p_V(v) = (1 - \eta)p_V(v)$$ \hspace{1cm} (15)

Then, we have the following joint probabilities:

$$p_{Z,W}(x,y) = \theta p_{X,Y}(x,y)$$ \hspace{1cm} (16)$$

$$p_{Z,W}(x,v) = \theta(1 - \eta)p_{X,V}(x,v)$$ \hspace{1cm} (17)$$

$$p_{Z,W}(u,y) = (1 - \theta)p_{U,Y}(u,y)$$ \hspace{1cm} (18)$$

$$p_{Z,W}(u,v) = (1 - \theta)(1 - \eta)p_{U,V}(u,v)$$ \hspace{1cm} (19)

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

$$I(X;Y) = \sum_x \sum_y p_{X,Y}(x,y) \log \frac{p_{X,Y}(x,y)}{p_X(x)p_Y(y)}$$ \hspace{1cm} (20)

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

$$I(Z;W) = \sum_{w} \sum_{z} p_{Z,W}(z,w) \log \frac{p_{Z,W}(z,w)}{p_Z(z)p_W(w)}$$ \hspace{1cm} (21)

Since $Z$ can take the values in $\mathcal{R}_Z$ if $z$ is drawn from $X$, and in $\mathcal{R}_U$ if $z$ is drawn from $U$ (similarly for $W$), then from Eq. (21), it follows that:

$$I(Z;W) = \sum_{y \in \mathcal{R}_Y} \sum_{x \in \mathcal{R}_X} p_{Z,W}(x,y) \log \frac{p_{Z,W}(x,y)}{p_Z(x)p_W(y)} + \sum_{u \in \mathcal{R}_U} \sum_{x \in \mathcal{R}_X} p_{Z,W}(x,u) \log \frac{p_{Z,W}(x,u)}{p_Z(x)p_W(u)} + \sum_{y \in \mathcal{R}_Y} \sum_{v \in \mathcal{R}_V} p_{Z,W}(x,v) \log \frac{p_{Z,W}(x,v)}{p_Z(x)p_W(v)} + \sum_{u \in \mathcal{R}_U} \sum_{v \in \mathcal{R}_V} p_{Z,W}(u,v) \log \frac{p_{Z,W}(u,v)}{p_Z(u)p_W(v)}$$

$$= \sum_{y \in \mathcal{R}_Y} \sum_{x \in \mathcal{R}_X} \theta p_{X,Y}(x,y) \log \frac{\theta p_{X,Y}(x,y)}{p_X(x)p_Y(y)} + \sum_{u \in \mathcal{R}_U} \sum_{x \in \mathcal{R}_X} \theta(1 - \eta)p_{X,V}(x,v) \log \frac{\theta(1 - \eta)p_{X,V}(x,v)}{p_X(x)(1 - \eta)p_V(v)} + \sum_{y \in \mathcal{R}_Y} \sum_{v \in \mathcal{R}_V} (1 - \theta)(1 - \eta)p_{U,V}(u,v) \log \frac{(1 - \theta)(1 - \eta)p_{U,V}(u,v)}{(1 - \theta)p_U(u)(1 - \eta)p_V(v)}$$ \hspace{1cm} (22)

Eq. (22) can be rewritten as

$$I(Z;W) = \theta I(X;Y) + (1 - \theta)\eta I(U;Y) + \theta(1 - \eta)I(X;V) + (1 - \theta)(1 - \eta)I(U;V)$$ \hspace{1cm} (23)

Since we assume: ($U \perp V$) $\land$ ($X \perp U$) $\land$ ($X \perp V$) $\land$ ($Y \perp U$) $\land$ ($Y \perp V$), this leads to: $I(U;Y) = 0$ and $I(X;V) = 0$ and $I(U;V) = 0$. Thus, Eq. (23) becomes

$$I(Z;W) = \theta I(X;Y)$$ \hspace{1cm} (24)

where $\theta \leq 1$ and $\eta \leq 1$. This leads to: $I(X;Y) \leq I(Z;W)$. \hspace{1cm} □

Theorem 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 $I(X;Y)$. Based on Theorem 1, we define noise as follows.

Definition 6.2 (Noise) Let $(X_T, Y_T)$ be a pair of time series, $\sigma$ be the correlation threshold, and $\tau (0 \leq \tau < \sigma)$ be the noise threshold. Consider two consecutive windows $w_{X,Y}$ and $w'_{X,Y}$ of $(X_T, Y_T)$, and $w''_{X,Y} = w_{X,Y} \oplus w'_{X,Y}$ is the mixture of $w_{X,Y}$ and $w'_{X,Y}$. Assume that $I_{w_{X,Y}} > 0$. Then $w_{X,Y}$ is called noise w.r.t. $w_{X,Y}$ iff $I_{w''_{X,Y}} < \tau \land I_{w''_{X,Y}} < I_{w_{X,Y}}$.

The following sections apply the noise identification principle in Theorem 1 and Def 6.2 to prune the search space of SYCOSTD and SYCOSBU, to speed up the search.

6.2 Noise-Based Pruning for SYCOSTD

Recall that SYCOSTD uses a sliding window technique to identify correlations in each layer. For example, consider the windows $w_0$, $w_1$, $w_2$ at $L_1$ in Fig. 2. Here, $w_0$ is an uncorrelated window, thus it is shifted by a $\delta$-step to form $w_1$, creating an overlapping data partition $[t_{s_1}, t_{e_1}]$ between $w_0$ and $w_1$. The next window $w_1$ is also uncorrelated, and is shifted to form $w_2$, creating the overlapping part $[t_{s_2}, t_{e_2}]$. By shifting one window to create another window, SYCOSTD creates overlapping data that will be revisited multiple times during the search process. To avoid multiple shiftings where overlapping data are repeatedly visited, the noise theory is applied to shorten this shifting step.

More specifically, when $w_1$ is created by shifting $w_0$ by a $\delta$-step, $w_1$ is divided into two sub-windows: $w_1 = w_0 \oplus w_\delta$ where $w_\delta = [t_{s_1}, t_{e_1}]$ represents the overlapping data, and $w_\delta = [t_{e_0}, t_{e_1}]$ represents the newly added data, and $\oplus$ is the concatenation operator. We apply the noise theory to $w_1$ and evaluate its goodness in 2 steps. First, $I_{w_1}$, $I_{w_0}$, and $I_{w_\delta}$ are computed separately. Next, $I_{w_1}$ and $I_{w_0}$ are compared against the correlation threshold $\sigma$, while $I_{w_\delta}$ is compared against the noise threshold $\tau$. Assume that by applying Theorem 1 we conclude that $w_\delta$ is noise w.r.t. $w_0$, i.e., $I_{w_\delta} < \tau \land I_{w_\delta} < I_{w_0}$, then noise is detected in $w_1$. In this case, instead of shifting $w_1$ to create $w_2$, the search can skip $w_1$ entirely, and move directly to $w_3$ whose start index $t_{s_3}$ is right after the end index $t_{e_1}$ of $w_1$, as in Fig. 2. In contrast, if noise is not detected in $w_1$, SYCOSTD shifts $w_1$ to $w_2$ as usual. By using the noise detection mechanism, the entire shifting process can be avoided, therefore speeding up the search performance.

However, to ensure the noise theory does not remove important data by chance, we require that noise should be consecutively detected $p$ times before canceling the shifting process. For instance, in the example above, after identifying that noise is present in $w_1$, SYCOSTD cancels the shifting process only if noise is also detected in the next $(p-1)$ consecutive time windows.

6.3 Noise-Based Pruning for SYCOSBU

Recall that SYCOSBU starts with an initial solution $w_0$, and tries to improve $w_0$ by searching for better solutions in nearby neighborhoods. To prune irrelevant data partitions and improve the search performance during the neighborhood exploration process, we apply the noise theory using the example in Fig. 3 as below.

Assume that $w$ is the current solution of SYCOSBU, and $w_{N_1}$, $w_{N_2}$ are its neighbors when the search moves leftwards. In the first exploration, the neighbor $w_{N_1}$ is considered. Since $w_{N_1}$ is created by extending the start index of $w$ by a $\delta_1$-step to the left, we have: $w_{N_1} = w_{\delta_1} \oplus w$ where $w_{\delta_1}$ is the extension to be added to $w$. Also assume that by applying the noise theory to $w, w_{\delta_1},$ and $w_{N_1}$, we conclude that $w_{\delta_1}$ is noise w.r.t. $w$, i.e., $I_{w_{\delta_1}} < \tau \land I_{w_{\delta_1}} < I_u$. In
this case, noise is detected in the leftwards direction of \( w \), and thus, it is not promising to explore further its leftwards neighborhoods. Hence, in the next exploration, SYCOS\(_{BU} \) does not consider \( w_{w_3} \) and omits the leftwards direction entirely from its search. Similarly to SYCOS\(_{TD} \), to avoid the risk of removing important data, we stop the search exploration towards a direction only after \( p \) noise detections. In the example above, after identifying that \( w_{w_3} \) is noise w.r.t. \( w \), SYCOS\(_{BU} \) only stops exploring leftwards after \( p \) noise detections.

### 7 Efficient and Distributed iSYCOS

In the previous section, we discussed how to integrate the noise theory into iSYCOS to prune irrelevant data partitions and reduce the search space of SYCOS\(_{TD} \) and SYCOS\(_{BU} \). In this section, we propose an efficient MI computation technique to reduce the computation redundancy across windows, further improving the search performance. Furthermore, we also propose a distributed version of iSYCOS that leverages the parallelism of a computing cluster using Apache Spark to search for correlations in big datasets.

#### 7.1 Efficient and Incremental MI Computation

##### 7.1.1 Box-assisted algorithm

Recall that to compute the MI of a window, the KSG estimator needs to estimate the population density of data points in that window. To do that, for each data point \( p_i = (x_i, y_i) \) in the considered window, it first searches for the \( k \) nearest neighbors of \( p_i \), and then counts the number of marginal points \( n_x, n_y \) that fall into the marginal \( k \)-nearest distances in each dimension \( d_x, d_y \). Among all the computations, \( k \)-nearest neighbor (\( knn \)) search is the most expensive operator. Therefore, we design an efficient data structure to optimize the \( knn \) search using the boxed-assisted method [49].

In the box-assisted method, the search space is divided into equal size boxes where each data point is projected into exactly one box. Each box maintains a list of points belonging to that box. When searching for the \( k \)-nearest neighbors of point \( p_i \), first the box containing \( p_i \) is found. Then, the search starts from that box and extends to nearby boxes to find the \( k \) nearest points. Next, the distances to the \( k \)th-neighbors in each dimension \( d_x, d_y \) are determined, and the marginal points \( n_x, n_y \) are computed by counting the number of points falling within these distances. The values of \( n_x \) and \( n_y \) will be used in Eq.2 to estimate the final MI.

Fig. 6a illustrates the boxed-assisted method and how it is used in KSG to compute the MI of a window that contains 7 data points \( p_1, \ldots, p_7 \). Consider point \( p_1 \) (red) in the figure. Assume that \( k = 2 \) is nearest neighbor parameter, and maximum norm \( \| \cdot \| \) is the distance metric. In this setting, the two nearest neighbors of \( p_1 \) are \( p_2 \) and \( p_3 \) (green), and the nearest distances in each dimension are \( d_x \) and \( d_y \) (shown in the figure). The nearest distances allow KSG 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 \( n_x = 3 \) (including \( p_2, p_3, p_5 \)), and \( n_y = 3 \) (including \( p_2, p_3, p_4 \)). A similar process is applied to other points in the window, and the final MI is computed by accumulating the obtained \( n_x \) and \( n_y \) values.

#### 7.1.2 Incremental MI computation with influenced regions

In Section 6, we show how the noise theory could avoid revisiting the same data partition multiple times. However, the MI computation across nearby windows can still be repeated. For example, in Fig. 6 when SYCOS\(_{BU} \) moves leftwards, it forms the neighbor window \( w_{w_1} = w_3 \odot w \) by adding a new data partition \( w_3 \) to the current window \( w \). Here, the shared data between \( w \) and \( w_{w_1} \) are exactly \( w \) itself. Similarly, in SYCOS\(_{TD} \), the shifting from \( w_1 \) to \( w_2 \) in Fig. 2 creates three different groups of data: group 1 from \( s_1 \) to \( s_2 \) contains the data points of \( w_1 \) to be removed from \( w_2 \), group 2 from \( s_2 \) to \( c_1 \) is overlapping data between \( w_1 \) and \( w_2 \), and group 3 from \( c_1 \) to \( c_2 \) is newly added data to \( w_2 \).

To reuse the computations on overlapping data across windows, we design an optimized boxed-assisted method, ensuring that new computation is only performed on new data, while on overlapping data, only those affected by changes (by removing or adding data) are re-evaluated. Note that the introduced changes can be either changing the \( k \)-nearest neighbors of existing points or changing the marginal counts \( n_x, n_y \). To keep track of the previous computation, we introduce the concepts of influenced region and influenced marginal region.

**Definition 7.1 (Influenced region)** An influenced region (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, which 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(s))** The influenced marginal region(s) (IMR) of point \( p_i \) is (are) the marginal region(s) located within the nearest distance \( d_i \) in each dimension.

Fig. 6b illustrates the influenced region and influenced marginal region concepts. The influenced region of \( p_i \) is the square colored in green, and the influenced marginal regions are those with gray shade in either dimension.

**Lemma 1.** Given a window \( w \) and a data point \( p \in w \), a new point \( o \) inserted into \( w \) will become the new \( k^{th} \) neighbor of \( p \) iff \( o \) is within IR of \( p \).

**Lemma 2.** Given a window \( w \) and a data point \( p \in w \), an existing point \( o \) deleted from \( w \) will change the \( k \) nearest points of \( p \) iff \( o \) is within IR of \( p \).
Lemma 3. 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 \( o \) if \( o \) is within IMR \( x \) (or IMR \( y \)) 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 reduce the marginal count \( n_x \) (or \( n_y \)) of \( o \) if \( o \) is within IMR \( x \) (or IMR \( y \)) of \( p \).

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

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

Fig. 6 illustrates how changes are introduced and managed. For simplicity, we only discuss the cases when new points are added into a previous computation. Changes introduced by removing points can be handled in a similar way. Assume that at time \( t_1 \), a new point \( p_b \) is added to the current window and falls into the IR of \( p_1 \). The addition of \( p_b \) changes the \( k \)-th nearest neighbor of \( p_1 \). Thus, a new nearest neighbor search for \( p_1 \) is required.

At time \( t_2 \), a new point \( p_b \) 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 \( p_1 \) 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 optimized box-assisted computation method, 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, thus minimizing the computational cost.

7.1.3 Time complexity analysis

The time complexities of SYCOS \(_{TD} \) and SYCOS \(_{BU} \) depend on the number of feasible windows \( n_w \) in the search space, and the time \( t_w \) to compute the MI value of each window.

Recall that the \( k \)-nearest neighbor (knn) search is the most expensive operator in the MI computation. Using the boxed-assisted method for efficient knn search, the time complexity of the MI computation is \( t_w \sim O(n \log n) \) where \( n \) is the window size. Moreover, we have the number of feasible windows \( n_w \sim O(N^2) \) for SYCOS \(_{TD} \), and \( n_w \sim O(N^2 T_{\text{maxdrive}}) \) for SYCOS \(_{BU} \).

Thus, the time complexity of SYCOS \(_{TD} \) is \( O(N^2 n \log n) \), and of SYCOS \(_{BU} \) is \( O(N^2 T_{\text{maxdrive}} n \log n) \), where \( n \) is the average window size.

However, we note that these are worst case complexities. In practice, thanks to the filtering of correlated windows during the search, and the incremental MI computation technique, the time complexities of SYCOS \(_{TD} \) and SYCOS \(_{BU} \) are much smaller (refers to the experiments in Section 8.5).

7.2 Distributed iSYCOS using Apache Spark

The distributed iSYCOS distributes the computation among worker nodes in a Spark cluster to speed up the search. This distributed computation applies to both the selection analysis (to select the most efficient search method among SYCOS\(_{BU} \) or SYCOS\(_{TD} \)), and the correlation search. Algorithm 3 provides the pseudo-code of the distributed iSYCOS, with the detailed explanation below.

In the selection analysis step (function \( \text{algSelection} \), lines 4–10), first, iSYCOS samples \( m \) data partitions from the input time series (lines 5–7). This data sampling step is performed at the master node since distribution is not needed for a fast operation. Next, iSYCOS maps each sampled data partition \( p_1 \) to a worker node (line 8), and executes SYCOS\(_{BU} \) and SYCOS\(_{TD} \) on \( p_1 \) (lines 1–3). The performance statistics obtained on \( p_1 \) are collected by Spark function \( \text{collect() \to \text{master node}} \). Finally, the efficiency scores \( n_{\text{score}} \) of SYCOS are computed at the master node (line 9), and a search method is selected using the policies in Section 5.3.

Using the selected search method, iSYCOS proceeds to distribute the correlation computation by first dividing the original time series into multiple overlapping data chunks (line 14). The overlapping data ensures the correlation analysis is contiguous between data chunks, and is equal to the maximum window size \( s_{\text{max}} \). Next, iSYCOS maps each overlapping data chunk to a worker node and invokes the selected algorithm, i.e., SYCOS\(_{BU} \) or SYCOS\(_{TD} \). SYCOS collects the extracted correlated windows from each worker node using the Spark function \( \text{collect()} \).

Algorithm 4 Distributed iSYCOS

Input: \( \{X_T, Y_T\} \): pair of time series variables  
\( s_{\text{max}} \): maximum window size  
\( n_s \): number of overlapping data chunks  
\( N \): length of time series  
\( \text{lenS} \): length of the sampled data partition  
\( \text{numS} \): number of the sampled data partitions

1: def algAnalysis(startIdx, endIdx):
2:     output ← subprocess.check_output(startIdx, endIdx)
3:     return output.decode()
4: def algSelection(N, lenS, numS):
5:     list_in_range ← [(x, x+lenS-1) for x in range(0, N, lenS)]
6:     samples ← random.sample(list_in_range, numS)
7:     samplesRDD ← sc.parallelize(samples)
8:     timestamps, W\(_{TD} \), W\(_{BU} \), W\(_{BL} \) ← samplesRDD.map(algAnalysis).collect()
9:     n_{\text{score}} \(_{TD} \), n_{\text{score}} \(_{BU} \) ← calculateScores(timestamps, W\(_{TD} \), W\(_{BU} \), W\(_{BL} \))
10: return n_{\text{score}} \(_{TD} \), n_{\text{score}} \(_{BU} \)
11: if _name_ == "_main_":
12:     sc ← SparkContext(conf=SparkConf());
13:     n_{\text{score}} \(_{TD} \), n_{\text{score}} \(_{BU} \) ← algSelection(N, lenS, numS)
14:     dataRDD ← sc.parallelize([i for i in xrange(0, N, n_s) for i])
15:     if n_{\text{score}} \(_{TD} \) > n_{\text{score}} \(_{BU} \) then:
16:         corrWindows ← dataRDD.map(SYCOS\(_{TD}\)).collect();
17:     else:
18:         corrWindows ← dataRDD.map(SYCOS\(_{BU}\)).collect();

8 Experimental Evaluation

In this section, we evaluate the performance of iSYCOS in terms of the quality of extracted windows, the runtime efficiency and the accuracy. We use both synthetic and real-world datasets from the energy and the smart city domains.
8.1 Baseline methods

**Qualitative evaluation:** To evaluate the quality of extracted windows, iSYCOS is compared against two baselines: (1) the traditional correlation metric Pearson Correlation Coefficient (PCC) [43,], and (2) Fast Subsequence Search (MASS) [43,], a well-known method for time series sub-sequences matching. We use MASS2 [2] which provides a faster implementation than the original MASS.

**Quantitative evaluation:** To be quantitatively comparable to iSYCOS, a baseline method must have similar objectives, i.e., it can detect correlations with multiple temporal scales. In this regard, we exclude seem-to-be-similar methods such as Data Polygamy [11] (because it does not consider multi-scale correlations), DTW [45] and MASS [43] (because they consider the problem of time series similarity search instead of correlation search). In fact, there are no existing methods that perform window-based correlation search as iSYCOS. For this reason, iSYCOS runtime efficiency is evaluated by comparing SYCOS_{BU} and SYCOS_{TD} against each other, using their different versions: with and without MI optimization and noise pruning. Furthermore, we also analyze the effectiveness of iSYCOS in selecting the most suitable method between SYCOS_{BU} and SYCOS_{TD} on different datasets.

8.2 Datasets

To test the robustness of iSYCOS, we generate synthetic datasets containing different types of relations, including both linear and non-linear, monotonic and non-monotonic, functional and non-functional relations. To examine iSYCOS effectiveness in identifying correlations in real-life scenarios, we use real-world data from the energy and the smart city domains. The smart city data are obtained from the NYC Open Data portal [3], containing more than 1,500 spatio-temporal datasets and providing a wide range of information about New York City. The energy data are obtained from an energy trading company in Belgium, under an NDA agreement, containing energy production data from multiple wind farms. Besides, we use another open energy dataset from Kaggle [11], providing energy consumption of electrical appliances in residential buildings in Belgium.

8.3 Parameter settings

iSYCOS depends on the setting of 4 parameters: correlation threshold $\sigma$, noise threshold $\varepsilon$, minimum window size $s_{\text{min}}$, and maximum window size $s_{\text{max}}$. Among them, $\sigma$, $s_{\text{min}}$, and $s_{\text{max}}$ are user-defined parameters, while $\varepsilon$ is a hyper parameter. The $\sigma$ value determines the strength of extracted correlations (larger $\sigma$, stronger correlations), while the value of $\varepsilon$ determines the degree of noise allowed over correlated data. For example, a fraction $\varepsilon/\sigma = 0.25$ means that data with MI less than $\frac{1}{4}$ of the correlation threshold are considered as noise and thus, unpromising to explore further. On the other hand, the values of $s_{\text{min}}$ and $s_{\text{max}}$ are context dependent, and are set based on domain knowledge. Given an application domain, it is usually intuitive how small or large a window could be. For example, when users analyze weather related data, they might decide that the longest duration of a weather event is two weeks, and thus set the size of $s_{\text{max}}$ to two weeks.

In our experiments, we use the normalized MI (scaled between $[0,1]$) introduced in Section 5.4 to set the value of $\sigma$. The hyper parameter $\varepsilon$ is set equal to $\frac{1}{4} \cdot \sigma$ in all experiments. The ratio $\varepsilon/\sigma = 0.25$ is chosen based on the empirical studies conducted on different datasets, which consistently show that $\varepsilon/\sigma \approx 0.25$ yields the best trade-off between accuracy and runtime gain. Table 1 lists the values of $\sigma$, $s_{\text{min}}$, and $s_{\text{max}}$ we use in each dataset.

| Parameter | Energy datasets | Smart city datasets |
|-----------|-----------------|---------------------|
| $\sigma$  | 0.2             | 0.2                 |
| $s_{\text{min}}$ | 30 samples $\pm$ 30 mins | 4 samples $\pm$ 1 hour |
| $s_{\text{max}}$ | 10,080 samples $\pm$ 7 days | 2,880 samples $\pm$ 30 days |

8.4 Qualitative evaluation

8.4.1 Using synthetic datasets

We apply both SYCOS_{BU} and SYCOS_{TD} to the synthetic data to test its ability to detect the generated relations. We compare the results to the baselines, i.e., PCC and MASS. Table 2 shows the types of relations, $y = f(x)$ and $u$ is added noise, recognized by each method.

Our experiments show that both methods SYCOS_{BU} and SYCOS_{TD} can identify all types of relations, even when there is noise present in the data. Compared to the baselines, PCC cannot detect non-functional relations, while MASS cannot identify non-linear and non-functional relations. To explain the weak performance of the baselines, we can see that MASS is designed to identify similarities in time series. Therefore, for non-linear and non-functional relations such as a circle, there is no similarity between a variable $x$ that is linearly increasing, and a variable $y$ that follows a round shape, and thus they cannot be identified by MASS.

### Table 2: Identify different types of correlation relations

| Relations: $y = f(x)$ | SYCOS_{BU} | SYCOS_{TD} | PCC | MASS |
|------------------------|------------|------------|-----|------|
| Ind.: $y \sim N(0,1)$, $x \sim N(3,5)$ | ✓ | ✓ | ✓ | ✓ |
| Linear: $y = 2x + u$, $x \in [0, 10]$ | ✓ | ✓ | ✓ | ✓ |
| Exponential: $y = 0.01x^2 + u$, $x \in [-10, 10]$ | ✓ | ✓ | ✓ | ✓ |
| Quadratic: $y = x^2 + u$, $x \in [-4, 4]$ | ✓ | ✓ | ✓ | ✓ |
| Diamond: $y_1 = x + u$, $y_2 = 8 - x + u$, $y_3 = -4 + x + u$, $y_4 = 12 - x + u$, $x \in [4, 8]$ | ✓ | ✓ | ✓ | ✓ |
| 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: $y = x^4 - 4x^3 + 4x^2 + x + u$, $x \in [-1, 1]$ | ✓ | ✓ | ✓ | ✓ |
| Square root: $y = \sqrt{x}$, $x \in [0, 25]$ | ✓ | ✓ | ✓ | ✓ |

*N(\mu, \sigma): normal distribution

$\sim U(0,1)$: uniform distribution
TABLE 3: Top K windows between Taxi and Wind Speed

| From       | To          | MI  | Event                |
|------------|-------------|-----|----------------------|
| 2012/10/29 | 2012/11/02  | 0,65| Sandy Hurricane      |
| 2012/07/27 | 2012/07/28  | 0,6 | Tornado hits NYC     |
| 2012/01/20 | 2012/01/21  | 0,57| Snow storm           |
| 2013/06/07 | 2013/06/10  | 0,54| Tropical storm Andrea|
| 2012/11/10 | 2012/11/11  | 0,54| Snow storm           |
| 2012/12/23 | 2012/12/24  | 0,49| Storm                |
| 2012/11/06 | 2012/11/07  | 0,48| Snow storm           |
| 2012/09/08 | 2012/09/09  | 0,42| Tornado              |
| 2011/08/26 | 2011/08/31  | 0,41| Irene Hurricane      |
| 2011/10/29 | 2011/10/31  | 0,40| Snow storm           |

8.4.2 Using real-world datasets

We apply SYCOS$_{BU}$ and SYCOS$_{TD}$ to real-world datasets to extract correlated windows. We interpret the insights from extracted correlations, and report a few of them below.

The smart city datasets: We focus on two collections of data related to transportation and weather. The data are measured in day, hour and minute resolutions.

We analyzed the correlations between the Weather and the Taxi datasets, and found a strong negative correlation (one variable increases while the other decreases) between the wind speed and the number of taxi trips in the city. When associating the correlated windows with their time intervals, we observe that the correlations between these two variables only occurred at times when extreme weather events happened in NYC, such as the occurrence of a hurricane or storm. We then ordered the extracted windows by their MI values, and found that many of the top ranked windows are in fact associated with extreme weather conditions, as reported in Table 3.

We test the relation between the average rain precipitation and the number of taxi trips. In the extracted windows, we found a negative correlation between these two variables: a drop in taxi trips associated with abnormally high rain. These correlations also occurred during extreme weather events, like the wind speed.

We test another pair of variables, the taxi fare and the rain precipitation. The authors in [11], when using the same datasets, report a positive relationship (both variables are increasing or decreasing together) between the fare taxi drivers earn and the rain precipitation, suggesting that taxi drivers increase earnings when it rains. In our findings, we also found this positive correlation. Moreover, we found that this correlation is visible only in hour resolution, but disappears in day resolution. This phenomena might be explained based on the fact that taxi drivers are target earners: taxi drivers have a daily income target, and reach their targets sooner when it rains, after which they quit driving for the remainder of the day, and thus keep the same daily income.

In another finding, we found a weak positive correlation between the taxi fare and the trip duration. The correlation suggests that taxi driver is likely to earn more when the trip duration is longer. Although this correlation seems obvious, it is not trivial why it is only a weak correlation. To explain this weak correlation, we note that trip duration gets longer either because of traffic jam (the fare does not increase) or because the travel distance is long (the fare does increase).

If traffic jam is the cause of long trip duration, linking trip duration to taxi fare is not enough, and thus, explains the weak correlation between the two variables. This suggests that other variables, such as travel distance, can be of interest to be analyzed together with taxi fare. The energy datasets: The energy data are recorded at high frequency (every 2 seconds), thus, we aggregate them into minute resolution before applying iSYCOS.

When analyzing the energy data, we first want to confirm that iSYCOS can find known correlations. Thus, we test the relation between the produced energy and its source, i.e., the active power and the solar irradiation, and the active power and the wind speed. The correlated windows extracted from iSYCOS indeed confirm the strong positive correlations between the produced energy and its respective source. For example, extracted windows show that the generated power increases when the solar irradiation is high or when the wind speed is strong, and vice versa. This strong correlation holds for the entire time series of the considered variables. Moreover, we found that the wind speed is strongly correlated with the rotor speed of the wind turbine.

When analyzing the relation between the produced power and weather-related variables, we found a negative correlation between the power production and the humidity, and a positive correlation between the power production and the temperature. This inspires us to test the relationship between the humidity and the temperature. As expected, we found a negative correlation between these two variables. These inter-correlations between the three variables are naturally intuitive, since high solar irradiation and low humidity result in high temperature, and vice versa. Similarly, when wind turbines are the energy source, we found a negative correlation between the air pressure and the wind speed, and thus, also a negative correlation between the air pressure and the produced power. These correlations can be explained based on meteorology, in which the air flows from high pressure area to low pressure area, and thus creates wind. This results in a negative correlation between the air pressure and the wind speed. Since the wind speed is directly correlated to the produced energy (through the wind turbine), it explains the negative correlation between the air pressure and the energy production.

A somewhat surprising finding is that we found the positive correlation between the wind direction and the wind speed. As wind direction indicates the direction in which the wind flows (measured in degrees), and wind speed describes how fast the air is moving (measured in km/hour), it is not trivial how to explain the nature of this correlation. However, the discovered correlation might prompt users new opportunities to perform further analysis. As we do not attempt to explain the causality behind the correlation, we refer readers to other articles such as [50] for further understanding of this phenomenon.

Similar meteorological phenomena are also found in the open energy dataset from Kaggle. For example, we confirmed a strongly negative correlation between the room temperature and the humidity, and between the humidity and the air pressure. In contrast, a strongly positive correlation is detected between the temperature and the air pressure.
8.5 Quantitative evaluation

8.5.1 SYCOS\textsubscript{TD} and SYCOS\textsubscript{BU} performance comparison

We first evaluate the runtime efficiency of iSYCOS by comparing the performance of SYCOS\textsubscript{TD} and SYCOS\textsubscript{BU} on different datasets. The experiments are run on a VM with 4 vCPUs, 32GB of RAM, and 1024GB of storage.

Fig. 7 shows the runtimes of SYCOS\textsubscript{TD} and SYCOS\textsubscript{BU}, and Table 4 shows the average MI and window size extracted from each method on different variable pairs of the smart city and the energy datasets. From Fig. 7, we can see that SYCOS\textsubscript{TD} and SYCOS\textsubscript{BU} perform differently on different datasets. There are cases where SYCOS\textsubscript{TD} outperforms SYCOS\textsubscript{BU}, and vice versa. For example, SYCOS\textsubscript{TD} outperforms SYCOS\textsubscript{BU} on the pairs (Taxi Trips, Traffic Speed), (Wind Speed, Active Power), (Rotor Speed, Active Power), and (Wind Speed, Rotor Speed). The average MI of extracted windows from these pairs are significantly higher than other pairs (e.g., 0.52-0.84), indicating that the variables are strongly correlated. The average window sizes of SYCOS\textsubscript{TD} are also significantly larger than SYCOS\textsubscript{BU}. This supports our initial hypothesis that the top-down approach is more efficient when the variables are strongly correlated, and the correlated windows are large. On these pairs of variables, SYCOS\textsubscript{TD} obtains an average speedup of 94.27 over SYCOS\textsubscript{BU}, and the speedup range is [1.4-370].

Conversely, SYCOS\textsubscript{BU} outperforms SYCOS\textsubscript{TD} on the pairs (Taxi Trips, Rain Precipitation), (Taxi Fare, Rain Precipitation), (Wind Direction, Pitch Angle), (Wind Direction, Wind Speed), and (Pitch Angle, Rotor Speed). It is noticeable that the average MIs of extracted windows from these pairs are significantly lower (e.g., 0.2-0.44) than those where SYCOS\textsubscript{TD} is faster, indicating that the variables are weakly or moderately correlated. The average sizes of extracted windows on these pairs are also significantly smaller. This also supports our initial hypothesis that the bottom-up approach is more efficient when the variables are not well correlated, and the correlated windows are small and sparse. On these pairs of variables, SYCOS\textsubscript{BU} obtains an average speedup of 5.3 over SYCOS\textsubscript{TD}, and the speedup range is [1.3-15.5]. We note that however, the average MIs of extracted windows from SYCOS\textsubscript{BU} and SYCOS\textsubscript{TD} are similar on each variable pair.

In summary, when comparing SYCOS\textsubscript{TD} and SYCOS\textsubscript{BU}, we conclude that SYCOS\textsubscript{TD} should be used to search for correlated windows when the variables are strongly correlated, while SYCOS\textsubscript{BU} should be used when the variables are weakly or moderately correlated.

8.5.2 Noise pruning and MI optimization

Next, we evaluate how effective the proposed optimizations, including the MI-based noise pruning (Section 6) and the MI optimization computation (Section 7.1), are in improving the performance of SYCOS\textsubscript{TD} and SYCOS\textsubscript{BU}. Particularly, we compare the runtimes of four different versions of SYCOS\textsubscript{TD} and SYCOS\textsubscript{BU} on different variable pairs: the original algorithm without optimizations (named as Origin), with noise theory (Noise), with MI optimization (MI Opt.), and with both noise theory and MI optimization (Both).

Figs. 8 and 9 show the results of this comparison. It can be seen that applying the proposed optimizations improves significantly the performance of both SYCOS\textsubscript{TD} and SYCOS\textsubscript{BU}. Specifically, the noise pruning and the MI optimization yield different speedups on the performance of the two methods. The noise pruning improves the performance of SYCOS\textsubscript{TD} by a speedup of 1.8 in average, and of SYCOS\textsubscript{BU} by a speedup of 7.4 in average. The MI optimized computation yields an average speedup of 6.7 on SYCOS\textsubscript{TD}, and an average speedup of 2.9 on SYCOS\textsubscript{BU}. However, applying both optimizations always yields better speedup, with an average of 15.5 on SYCOS\textsubscript{TD}, and 17.8 on SYCOS\textsubscript{BU}.

In Figs. 8 and 9, we observe that the noise pruning is more efficient than the MI optimized computation on the pairs (Taxi, Traffic Speed), (Taxi, Rain), (Wind Direction, Wind Speed), and (Pitch Angle, Rotor Speed), and less efficient on the pairs (Wind Speed, Active Power), (Rotor Speed, Active Power), and (Wind Speed, Rotor Speed). This indicates that the effectiveness of the proposed optimizations is data dependent. However, a general trend is that, the noise pruning is more efficient than the MI optimization when the data are weakly correlated (thus, more noise is present in the data) as shown in Figs. 8a, 8c, and 8d. In contrast, the MI optimization is more efficient when the data are strongly correlated (thus, shared MI computation occurs more often), as shown in Figs. 8b, 8c, and 8d. This coincides with the way SYCOS\textsubscript{TD} and SYCOS\textsubscript{BU} operate, where we observe that the noise pruning is generally more efficient in SYCOS\textsubscript{BU} than in SYCOS\textsubscript{TD}, and the MI optimization is more efficient in SYCOS\textsubscript{TD} than in SYCOS\textsubscript{BU}.

8.5.3 The accuracy of noise pruning

Next, we evaluate how accurate the noise pruning is compared to the exhaustive search. Particularly, we compare the similarity of windows extracted from SYCOS\textsubscript{TD} and SYCOS\textsubscript{BU} with and without noise pruning. Note that two windows are considered to be similar if they cover a similar range of data, i.e., the indices of the two windows are overlapping. Table 5 shows the accuracy of SYCOS\textsubscript{TD} and SYCOS\textsubscript{BU} on the tested datasets. It can be seen that the noise pruning yields highly accurate results, with the accuracy ranging from 80% to 100%.

8.5.4 Effectiveness of the selection algorithm

We evaluate the effectiveness of iSYCOS in selecting the most efficient search method for a given pair of input

| Data                   | Method | Avg. W. Size | Avg. MI |
|------------------------|--------|--------------|---------|
| Taxi Trips, Traffic Speed | TD     | 719          | 0.84    |
|                        | BU     | 22           | 0.68    |
| Taxi Trips, Rain       | TD     | 24           | 0.25    |
|                        | BU     | 14           | 0.35    |
| Taxi Fare, Rain        | TD     | 19           | 0.36    |
|                        | BU     | 16           | 0.44    |
| Wind Speed, A. Power   | TD     | 549          | 0.6     |
|                        | BU     | 184          | 0.62    |
| R. Speed, A. Power     | TD     | 662          | 0.68    |
|                        | BU     | 116          | 0.64    |
| Wind Speed, Rotor Speed| TD     | 648          | 0.57    |
|                        | BU     | 131          | 0.52    |
| W. Direction, Pitch Angle | TD     | 47           | 0.24    |
|                         | BU     | 16           | 0.28    |
| W. Direction, W. Speed | TD     | 86           | 0.35    |
|                         | BU     | 56           | 0.37    |
time series. More specifically, for each pair of variables, we perform random sampling to select 6 data partitions, and apply the selection algorithm (Section 5.3) on them. Table 6 ($\alpha = 0.5$) shows the computed efficiency scores of SYCOS$_{TD}$ and SYCOS$_{BU}$. The method with higher score (in bold) is chosen. As can be seen, iSYCOS selects SYCOS$_{TD}$ for the pairs (Taxi Trips, Traffic Speed), (Wind Speed, Active Power), (Rotor Speed, Active Power), and SYCOS$_{BU}$ for the rest. Compared to the analysis in Section 8.5.1, iSYCOS has correctly selected the most efficient search approach for each tested variable pair.

Notice however that iSYCOS has selected SYCOS$_{BU}$ instead of SYCOS$_{TD}$ for the pair (Wind Speed, Rotor Speed). When we further analyze this pair, we observe that the runtimes of SYCOS$_{TD}$ or SYCOS$_{BU}$ on this pair are similar (Fig. 7) on the sampled data partitions. Thus, their efficiency scores are close. However, as we change the weight $\alpha$ to $\alpha = 0.2$ (put more weight on the large window than on the runtime), we obtain the efficiency scores $n_{score_{BU}} = 0.79$ and $n_{score_{TD}} = 0.86$. In this case, iSYCOS selects SYCOS$_{TD}$.
Next, we analyze the impact of \( \alpha \) on iSYCOS selection by changing \( \alpha \) in the range \((0.2, \ldots, 0.8)\), and computing the efficiency scores of SYCOS\(_{TD}\) and SYCOS\(_{BU}\) accordingly. We find that when the data are strongly or weakly correlated, iSYCOS selection is robust w.r.t. \( \alpha \): changing \( \alpha \) does not change the selection. This is intuitive as when SYCOS\(_{TD}\) is significantly better than SYCOS\(_{BU}\) (strong correlation) or vice versa (weak correlation), their efficiency scores cannot be changed by \( \alpha \) in a way that alters the selection. However, in cases where data are moderately correlated, and SYCOS\(_{TD}\) and SYCOS\(_{BU}\) performances become comparable such as the case of (Wind Speed, Rotor Speed), changing \( \alpha \) can change the selection between SYCOS\(_{TD}\) and SYCOS\(_{BU}\), depending on whether the runtime or the window size is weighted more.

### 8.5.5 iSYCOS scalability

Finally, we evaluate the scalability of iSYCOS on large synthetic datasets generated from the smart city and the energy data. The synthetic data contain up to 100 millions of data points. We use a Spark cluster containing 20 high-end nodes, each with 64 cores, 256GB of RAM, 24TB of storage, and runs Apache Spark 3.1.

On the Spark cluster, we fix the data size to 100M data points while changing the number of worker nodes from 1 to 20. Fig. 10 shows the runtimes of the distributed iSYCOS on different datasets, as well as the speedup obtained when the computing resources increase. It can be seen that, iSYCOS scales well on the large datasets, and obtain a linear speedup w.r.t. the available resources.

### 9 Conclusion and Future Work

In the present paper, we propose an integrated SYNchronous COrelation Search (iSYCOS) framework to find multi-scale temporal correlations in big time series. iSYCOS integrates both the top-down approach SYCOS\(_{TD}\) and the bottom-up approach SYCOS\(_{BU}\) into a single framework, and is thus able to efficiently extract various types of correlation relations, including both linear and non-linear, monotonic and non-monotonic, functional and non-functional, from time series. Our main contributions are the two search approaches that fit to specific scenarios (top-down for strongly correlated
data, and bottom-up for weakly correlated data), and the proposed novel MI-based theory to identify noise in the data, and the MI optimized computation technique to optimize the search process. We also propose a robust selection algorithm that can automatically analyze and select the most efficient search approach for a given input time series. We further design a distributed version of iSYCOS that scales well on a Spark cluster to handle big time series. We perform an extensive experimental evaluation to evaluate the effectiveness and the performance of iSYCOS using both synthetic and real-world datasets. The evaluation shows that our method can capture various types of relations in synthetic data, and interesting correlations in real-world data. Moreover, the proposed optimizations are shown to be very efficient, and result in a speedup of up to an order of magnitude. The selection algorithm is robust on different datasets, and the distributed iSYCOS scales linearly to the data size and the number of computing nodes.

In future work, iSYCOS will be extended to capture correlations across spatial dimensions. The result of this work can also provide the foundation for advanced data analytics, such as to perform mining or infer causal effects from the extracted correlations.

REFERENCES

[1] Appliances Energy Prediction. https://www.kaggle.com/loveall/appliances-energy-prediction.

[2] mass-ts. https://github.com/matrix-profile-foundation/mass-ts.

[3] NYC Open Data. https://opendata.cityofnewyork.us.

[4] A. Agresti and B. Finlay. Statistical Methods for the Social Sciences. Pearson Education Limited, 2014.

[5] A. Alawini, D. Maier, K. Tufte, and B. Howe. Helping scientists reconnect their datasets. In SSDBM Proc., 2014.

[6] A. Barkat, A. D. dos Santos, and T. T. N. Ho. Open stack and cloud stack: Open source solutions for building public and private clouds. In 2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, pages 429–436. IEEE, 2014.

[7] E. K. Burke and Y. Bykov. The late acceptance hill-climbing heuristic. European Journal of Operational Research, 258(1):70–78, 2017.

[8] Q. Cai, Z. Xie, M. Zhang, G. Chen, H. Jagadish, and B. C. Ooi. Effective temporal dependence discovery in time series data. VLDB, 11(8):893–905, 2015.

[9] A. Camera, J. Shihe, T. Palpanas, T. Rakhmanonan, and E. Keogh. Beyond one billion time series: indexing and mining very large time series collections with isa2+. KIABS, 39(1):125–151, 2014.

[10] C. Cappiello, N. T. T. Ho, B. Pernici, P. Plebani, and M. Vitali. Co 2-aware adaptation strategies for cloud applications. IEEE Transactions on Cloud Computing, 4(2):152–165, 2015.

[11] F. Chirigati, H. Doraiswamy, T. Damoulas, and J. Freire. Data polygamy: the many-many relationships among urban spatio-temporal data sets. In SIGMOD Proc., 2016.

[12] R. Cole, D. Shasha, and X. Zhao. Fast window correlations over uncooperative time series. In ACM SIGKDD, pages 743–749. ACM, 2005.

[13] T. M. Cover and J. A. Thomas. Elements of information theory. John Wiley & Sons, 2012.

[14] A. Das Sarma, L. Fang, N. Gupta, A. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu. Finding related tables. In SIGMOD Proc., pages 817–828, 2012.

[15] S. de Siqueira Santos, D. Y. Takahashi, A. Nakata, and A. Fujita. A comparative study of statistical methods used to identify dependencies between gene expression signals. Briefings in bioinformatics, 15(6):906–918, 2013.

[16] E. P. de Sousa, C. Traina Jr, A. J. Traina, L. Wu, and C. Faloutsos. A fast and effective method to find correlations among attributes in databases. Data Mining and Knowledge Discovery, 14(3):367–407, 2007.

[17] W. E. Dean Jr and R. Y. Anderson. Application of some correlation coefficient techniques to time-series analysis. Journal of the International Association for Mathematical Geology, 6(4):363–372, 1974.

[18] P. A. Estévez, M. Tesmer, C. A. Perez, and J. M. Zurada. Normalized mutual information feature selection. TNN, 20(2):189–201, 2009.

[19] M. Gribaudo, T. T. N. Ho, B. Pernici, and G. Serazzi. Analysis of the influence of application deployment on energy consumption. In International Workshop on Energy Efficient Data Centers, pages 87–101. Springer, 2014.

[20] N. Ho, M. Gribaudo, and B. Pernici. Improving energy efficiency for transactional workloads in cloud environments. In Proceedings of the Eighth International Conference on Future Energy Systems, pages 290–295, 2017.

[21] N. Ho, T. B. Pedersen, V. L. Ho, and M. Vu. Efficient search for multi-scale time delay correlations in big time series data. In EDBT, pages 37–48, 2020.

[22] N. Ho, T. B. Pedersen, M. Vu, C. A. Biscio, et al. Efficient bottom-up discovery of multi-scale time series correlations using mutual information. In IEEE ICDE, pages 1734–1737. IEEE, 2019.

[23] N. Ho, T. B. Pedersen, M. Vu, et al. Efficient and distributed temporal pattern mining. In 2021 IEEE International Conference on Big Data (Big Data), pages 335–343. IEEE, 2021.

[24] N. Ho, H. Vo, and M. Vu. An adaptive information-theoretic approach for identifying temporal correlations in big data sets. In IEEE Big Data, pages 666–675. IEEE, 2016.

[25] N. T. T. Ho, H. Vo, M. Vu, and T. B. Pedersen. Amic: An adaptive information theoretic method to identify multi-scale temporal correlations in big time series data. IEEE Transactions on Big Data, 2019.

[26] T. T. N. Ho. Activity recognition using smartphone based sensors. 2013.

[27] T. T. N. HO. Towards sustainable solutions for applications in cloud computing and big data. 2017.

[28] T. T. N. Ho, M. Gribaudo, and B. Pernici. Characterizing energy per job in cloud applications. Electronics, 5(4):90, 2016.

[29] T. T. N. Ho and B. Pernici. A data-value-driven adaptation framework for energy efficiency for data
intensive applications in clouds. In 2015 IEEE conference on technologies for sustainability (SusTech), pages 47–52. IEEE, 2015.

[30] V. L. Ho, N. Ho, and T. B. Pedersen. Efficient temporal pattern mining in big time series using mutual information–full version. *arXiv preprint arXiv:2010.03653*, 2020.

[31] V. L. Ho, N. Ho, and T. B. Pedersen. Efficient temporal pattern mining in big time series using mutual information–full version. *arXiv preprint arXiv:2010.03653*, 2020.

[32] H.-C. Huang, S. Zheng, and Z. Zhao. Application of pearson correlation coefficient (pcc) and kolmogorov-smirnov distance (ksd) metrics to identify disease-specific biomarker genes. *BMC Bioinformatics*, 11(4):1, 2010.

[33] Y. Ke, J. Cheng, and W. Ng. An information-theoretic approach to quantitative association rule mining. *KAIS*, 16(2):213–244, 2008.

[34] F. Keller, E. Müller, and K. Böhm. Estimating mutual information on data streams. In *SSDBM Proc.*, 2015.

[35] J. F. Kenney. *Mathematics of statistics*. D. Van Nostrand Company Inc; Toronto; Princeton; New Jersey; London; New . . . , 2013.

[36] A. Kraskov, H. Stögbauer, and P. Grassberger. Estimating mutual information. *Physical review E*, 69(6):066138, 2004.

[37] P. E. Meyer, K. Kontos, F. Lafitte, and G. Bontempi. Information-theoretic inference of large transcriptional regulatory networks. *EURASIP journal on bioinformatics and systems biology*, 2007(1):1–9, 2007.

[38] M. Middelfart, T. B. Pedersen, and J. Krogsgaard. Efficient sentinel mining using bitmaps on modern processors. *TKDE*, 25(10):2231–2244, 2013.

[39] A. Papana and D. Kugiumtzis. Evaluation of mutual information estimators for time series. *International Journal of Bifurcation and Chaos*, 19(12):4197–4215, 2009.

[40] K. Pearson. Notes on regression and inheritance in the case of two parents, 1895.

[41] J. P. Pluim, J. A. Maintz, and M. A. Viergever. Mutual-information-based registration of medical images: a survey. *IEEE Trans. on Medical Imaging*, 22(8):986–1004, 2003.

[42] R. Pochampally, A. Das Sarma, X. L. Dong, A. Meliou, and D. Srivastava. Fusing data with correlations. In *SIGMOD Proc.*, 2014.

[43] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *ACM SIGKDD*, 2012.

[44] S. Roy and D. Suciu. A formal approach to finding explanations for database queries. In *SIGMOD Proc.*, 2014.

[45] S. J. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited., 2016.

[46] S. Salvador and P. Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561–580, 2007.

[47] D. Schulz and J. P. Huston. The sliding window correlation procedure for detecting hidden correlations: existence of behavioral subgroups illustrated with aged rats. *Journal of neuroscience methods*, 121(2):129–137, 2002.

[48] Y. Su, G. Agrawal, J. Woodring, A. Biswas, and H.-W. Shen. Supporting correlation analysis on scientific datasets in parallel and distributed settings. In *HPDC Proc.*, 2014.

[49] M. Vejmelka and K. Hlaváčková-Schindler. Mutual information estimation in higher dimensions: A speed-up of a k-nearest neighbor based estimator. In *ICANNGA*, pages 790–797. Springer, 2007.

[50] S. Velázquez, J. A. Carta, and J. Mattás. Comparison between anns and linear mcp algorithms in the long-term estimation of the cost per kwh produced by a wind turbine at a candidate site: a case study in the canary islands. *Applied energy*, 88(11):3869–3881, 2011.

[51] Q. Xie, S. Shang, B. Yuan, C. Pang, and X. Zhang. Local correlation detection with linearity enhancement in streaming data. In *CIKM*, pages 309–318. ACM, 2013.

[52] J. Yang, W. Wang, H. Wang, and P. Yu. δ-clusters: Capturing subspace correlation in a large data set. In *Data Engineering Proc.*, 2002.

[53] P. Zhang, Y. Huang, S. Shekhar, and V. Kumar. Correlation analysis of spatial time series datasets: A filter-and-refine approach. In *PAKDD Proc.*, 2003.