Assembler: Efficient Discovery of Spatial Co-evolving Patterns in Massive Geo-sensory Data

Chao Zhang[1*] Yu Zheng[2] Xiuli Ma[3] Jiawei Han[1]
[1]Dept. of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA
[2]Microsoft Research, Beijing, China
[3]School of Electronics Engineering and Computer Science, Peking University, Beijing, China
[1]{czhang82, hanj}@illinois.edu [2]yuzheng@microsoft.com [3]maxl@cis.pku.edu.cn

ABSTRACT

Recent years have witnessed the wide proliferation of geo-sensory applications wherein a bundle of sensors are deployed at different locations to cooperatively monitor the target condition. Given massive geo-sensory data, we study the problem of mining spatial co-evolving patterns (SCPs), *i.e.*, groups of sensors that are spatially correlated and co-evolve frequently in their readings. SCP mining is of great importance to various real-world applications, yet it is challenging because (1) the truly interesting evolutions are often flooded by numerous trivial fluctuations in the geo-sensory time series; and (2) the pattern search space is extremely large due to the spatiotemporal combinatorial nature of SCP. In this paper, we propose a two-stage method called `Assembler`. In the first stage, `Assembler` filters trivial fluctuations using wavelet transform and detects frequent evolutions for individual sensors via a segment-and-group approach. In the second stage, `Assembler` generates SCPs by assembling the frequent evolutions of individual sensors. Specifically, it leverages the spatial constraint to conceptually organize all the SCPs into a novel structure called the SCP search tree, which facilitates the effective pruning of the search space to generate SCPs efficiently. Our experiments on both real and synthetic data sets show that `Assembler` is effective, efficient, and scalable.

1. INTRODUCTION

Wireless sensor network (WSN) has been serving as an indispensable tool for our society ever since its advent in the 1970s. In a typical WSN, a bunch of sensors are deployed at different geographical locations to continuously and cooperatively monitor the target condition (*e.g.*, air pollution, temperature, traffic volume). Spurred by the recent development of urban computing [24], modern geo-sensory applications, ranging from traffic monitoring to environment control, often involve hundreds of sensors, and each sensor can potentially generate millions of records — depending on the sampling rate and duration.

While the massive geo-sensory data can be informative multi-facetedly, one particularly important and challenging problem is, how to discover a group of sensors that are spatially correlated and co-evolve frequently in their readings? Let us consider a real-life scenario in Figure 1. As shown, a number of air quality sensors are deployed in Beijing, and every hour, each sensor measures the real-time air quality index (AQI[1]) around it. The AQI measured by each sensor is stable most of the time, but does change substantially during some time intervals — which we call *evolving intervals*. During the three evolving intervals marked in Figure 1, one can observe that sensor s_1, s_2, and s_3 exhibit a recurring behavior: the AQIs of s_2 and s_3 increase substantially, while the AQI of s_1 drops sharply. Since s_1, s_2, and s_3 are spatially close, it is likely the air pollution is dispersed from s_1 to s_2 and s_3. According to an investigation into this pattern, the road traffic largely flows from s_1 to $\{s_2, s_3\}$ during off-work hours, deteriorating the air quality there with heavy emissions.

Given a set $\mathcal{S}$ of sensors and their records over n timestamps, we are interested in discovering groups of spatially correlated sensors from $\mathcal{S}$, along with their co-evolving behaviors that occur frequently in the n timestamps. We call such frequent behaviors *spatial co-evolving patterns* (SCPs for short). The discovery of SCPs has significant impacts on a wide spectrum of applications. Continuing with the above air quality monitoring example, the pattern $s_1 \rightarrow \{s_2, s_3\}$ indicates that air pollution is frequently dispersed from s_1 to s_2 and s_3. By analyzing the contexts (*e.g.*, traffic, weather, social activities) in which the dispersions occur, we can identify the major factors that cause such behaviors and come up with effective strategies to alleviate air pollutant diffusion throughout the city. Another example application is traffic analysis. Assume we have deployed sensors to monitor the traffic volumes in different regions within a city. If severe traffic jams are often observed in a region r, the SCPs among different regions can help us discover the regions from which the traffic tends to flow into r and cause the traffic jam. Such an understanding is very useful for improving the road network design and urban planning.

Considering the sheer size of modern geo-sensory systems, mining all SCPs from massive geo-sensory data is by no means a trivial problem. *First, interesting evolutions are often flooded by trivial fluctuations*. In practice, each sensor constantly generates measurements no matter the condition changes or stays stable, hence the time series is overwhelmed by numerous uninteresting fluctuations. For example, the histogram in Figure 2(a) is obtained from one-year AQI data of the sensor s_1 in Figure 1, by computing the AQI change of every pair of consecutive measurements. We can see the change is highly concentrated on small values. While many of the small values result from random fluctuations and redundant sampling, some are caused by slow but long-lasting evolutions (Figure 2(b)). It is important yet challenging to filter uninteresting fluctu-

*The research was done when the first author was an intern in Microsoft Research under the supervision of the second author.

KDD'15, August 10-13, 2015, Sydney, NSW, Australia.
 ISBN 978-1-4503-3664-2/15/08 ...$15.00.
DOI: http://dx.doi.org/10.1145/2783258.2783394.

[1]Larger AQI means more air pollutants and thus worse air quality.

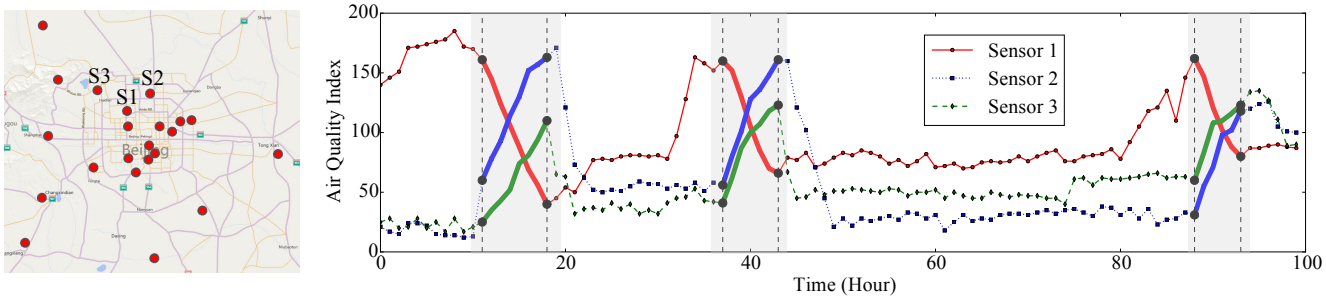
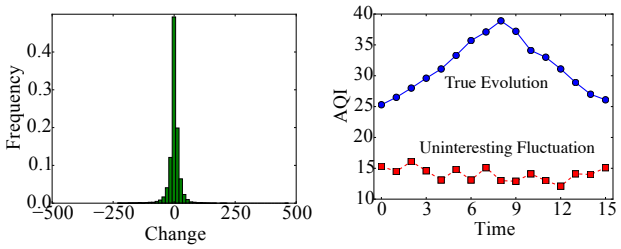


Figure 1: The air quality sensors in Beijing and the measurements of sensor s_1, s_2 and s_3 during an 100-hour period. The gray areas mark three evolving intervals during which the measurements of s_1, s_2, and s_3 change significantly.

ations and identify evolving intervals in the time series. *Second, the pattern search space is extremely large*. An SCP can contain an arbitrary number of sensors, and the matching time intervals of an SCP can have quite different durations. The combinatorial nature of SCP leads to an exponential pattern search space, calling for novel and efficient pattern search methodologies.



(a) Change distribution. (b) Evolution *v.s.* fluctuation.

Figure 2: Characteristics of geo-sensory time series.

Although multi-dimensional motif discovery techniques [20, 14, 17] have been developed to retrieve recurring subsequences in a bundle of time series, none of them are applicable to SCP mining. First, since geo-sensory data typically suffers from redundant sampling, applying motif discovery can only retrieve trivial motifs where the condition shows little change. Second, the matching subsequences of a motif must have the same length, whereas the matching intervals of an SCP usually differ in length due to the duration variation of the pattern. Last, motif discovery overlooks the spatial correlations among the sensors and typically obtain patterns that span across all the dimensions. In contrast, SCP mining requires one to discover groups of sub-dimensions (*i.e.*, sensors) that are spatially correlated and co-evolve frequently.

In this paper, we present `Assembler`, a two-stage method that can effectively and efficiently discover SCPs from massive geo-sensory data. `Assembler` first extracts frequent evolutions for each individual sensor, and then finds SCPs by merging the frequent evolutions of individual sensors. Our main contributions are summarized as follows.

1. In the first stage of `Assembler` (Section 3.1), we decompose each geo-sensory time series using *wavelet transform*, and show that different evolutions are captured by the large wavelet coefficients at different levels. Hence, we filter the trivial fluctuations in the wavelet space, and design a *segment-and-group* approach to effectively extract frequent evolutions for individual sensors.

2. In the second stage of `Assembler` (Section 3.2), we prove SCP satisfies the *anti-monotonicity* property. Although the Apriori rule [1] can be used to prune the search space and find all SCPs, it overlooks the spatial constraint and also suffers from excessive generation of candidate patterns. We design a novel structure called the *SCP search tree*, which conceptually organizes all the SCPs based on the spatial constraint. The depth-first construction of the SCP search tree can efficiently retrieve all SCPs without candidate generation.

3. We have conducted extensive experiments on both real-life and synthetic geo-sensory data sets (Section 4). The results demonstrate that `Assembler` is indeed capable of retrieving meaningful patterns, and it achieves excellent efficiency and scalability.

2. PROBLEM DESCRIPTION

Let $\mathcal{S} = \{s_1, s_2, \ldots, s_m\}$ be a set of sensors in a geographical region. Each sensor $s_i \in \mathcal{S}$ $(1 \leq i \leq m)$ is deployed at a location l_i to periodically measure the target condition around it. All the sensors in $\mathcal{S}$ have synchronized measurements over the time domain $\mathcal{T} = \langle t_1, t_2, \ldots, t_n \rangle$, where each t_j $(1 \leq j \leq n)$ is a timestamp and all t_j's are equally spaced. The measurement of sensor s_i at timestamp t_j, denoted by $s_i[t_j]$, is a real value.

As mentioned earlier, the time series of each sensor often suffers from redundant sampling, *i.e.*, the time domain $\mathcal{T}$ includes many trivial intervals in which the monitored condition shows random and small fluctuations. To obtain meaningful patterns, we are not interested in the entire time domain $\mathcal{T}$, but only demand the intervals in which the condition exhibits evident changes.

DEFINITION 1. *[Evolving Interval] For sensor s_i, a length-l evolving interval $\mathcal{I} = \langle t_j, t_{j+1}, \ldots, t_{j+l} \rangle$ is a consecutive subsequence in $\mathcal{T}$ where the measurement of s_i has evident change. The timestamps $t_j, \ldots, t_{j+l-1}$ are called evolving timestamps for s_i.*

DEFINITION 2. *[Change Rate] Given a sensor $s_i \in \mathcal{S}$ and an evolving timestamp t_j, the change rate of s_i at timestamp t_j is*

$$r_i[t_j] = \frac{s_i[t_{j+1}] - s_i[t_j]}{t_{j+1} - t_j}.$$

Note that: (1) Definition 1 provides an informal and intuitive description of evolving interval. Later in Section 3.1.1, we will elaborate this definition and show how we define "evident" changes. (2) Each sensor can have multiple evolving intervals, and the evolving intervals of two different sensors do not necessarily overlap.

One may wonder why not just examine the change rate for every timestamp in $\mathcal{T}$ and extract the timestamps having large change rates to define evolving intervals. The reason is that such a definition tends to miss slow but long-lasting changes: as shown by the example in Figure 3, while both slow and sharp changes are interesting evolutions, change rate alone only identifies sharp changes and fails to distinguish slow changes from trivial fluctuations.

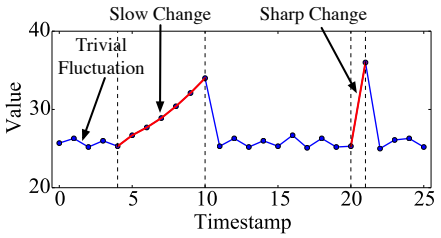


Figure 3: Trivial fluctuation, slow change, and sharp change in a geo-sensory time series: (1) in $[t_0, t_4]$, the measure shows random and trivial fluctuations; (2) in $[t_4, t_{11}]$, the measure shows slow but long-lasting increase; and (3) in $[t_{20}, t_{21}]$, the measure increases sharply.

Given the set $\mathcal{S}$ of sensors, our goal is to discover a group of sensors from $\mathcal{S}$ that are spatially correlated and meanwhile exhibit frequent co-evolutions. We define *spatial connectivity* and *co-evolution* as follows.

DEFINITION 3. *[Spatial Connectivity] Given a distance threshold h and a subset of sensors $G \subseteq \mathcal{S}$, G is spatially connected if $\forall s \in G, \exists s' \in G - \{s\}$ s.t. $\texttt{dist}(s, s') \leq h$ where $\texttt{dist}(s, s')$ is the geographical distance between s and s'.*

DEFINITION 4. *[Co-evolution] Let $G = \{s_1, s_2, \ldots, s_g\}$ be a set of spatially connected sensors. A co-evolution E_G over G has the form $\{R_1, R_2, \ldots, R_g\}$ where each $R_i = [lb_i, ub_i]$ $(1 \leq i \leq g)$ specifies a range of change rate for sensor $s_i \in G$.*

Now we proceed to define the matching relationship between a co-evolution and a timestamp, based on which we define the *support* of a co-evolution.

DEFINITION 5 (MATCH). *Let $G = \{s_1, s_2, \ldots, s_g\}$ be a set of spatially connected sensors and $E_G = \{R_1, R_2, \ldots, R_g\}$ be a co-evolution on G. A timestamp t_j matches E_G, denoted as $t_j \rightsquigarrow E_G$, if t_j satisfies $\forall 1 \leq i \leq g$: (1) t_j is an evolving timestamp for sensor s_i; and (2) the change rate of s_i at t_j falls in the range $R_i = [lb_i, ub_i]$, namely $lb_i \leq r_i[t_j] \leq ub_i$.*

DEFINITION 6 (SUPPORT). *The support of a co-evolution E_G is the number of its matching timestamps in the time domain $\mathcal{T}$,* i.e.*, $Sup(E_G) = |\{t_j | t_j \in \mathcal{T} \wedge t_j \rightsquigarrow E_G\}|$.*

If a co-evolution appears frequently in the time domain, we call it a *spatial co-evolving pattern* (SCP).

DEFINITION 7 (SPATIAL CO-EVOLVING PATTERN). *Let E_G be a co-evolution defined over a set G of spatially connected sensors. Given a minimum support θ, E_G is a spatial co-evolving pattern if $Sup(E_G) \geq \theta$.*

EXAMPLE 1. *In Figure 1, let $G = \{s_1, s_2, s_3\}$ be a set of spatially connected sensors. An example co-evolution over G is*

$$E_G = \{[-20/h, -15/h], [+15/h, +20/h], [+15/h, +20/h]\},$$

meaning the AQI of s_1 is decreasing by 15 to 20 per hour, while the AQIs of s_2 and s_3 are increasing by 15 to 20 per hour. Given a minimum support $\theta = 10$, E_G is a frequent pattern as it matches 24 evolving timestamps in the 100-hour period.

We are now ready to describe the SCP mining problem. Given the sensory data of $\mathcal{S}$ over the time domain $\mathcal{T}$, a minimum support θ, and a distance threshold h, find the groups of spatially connected sensors from $\mathcal{S}$ along with their co-evolutions that have support larger than θ.

3. THE ASSEMBLER METHOD

In this section, we present our two-stage SCP mining method `Assembler`. In what follows, we describe the detailed two stages in Section 3.1 and 3.2, respectively. Then we analyze the cost of `Assembler` and discuss how to set the parameters in Section 3.3.

3.1 Stage I: Detecting Individual Evolutions

In this subsection, we present the first stage of `Assembler`. It first extracts evolving intervals using wavelet transform, and then detects frequent evolutions for individual sensors via a segment-and-group approach.

3.1.1 Evolving Interval Extraction

As mentioned earlier, geo-sensory time series is usually overwhelmed by numerous trivial fluctuations due to redundant sampling. To find meaningful SCPs, one first needs to filter uninteresting fluctuations and identify the evolving intervals, otherwise the result pattern set will be dominated by the trivial fluctuations. However, the condition change can occur with quite different rates and durations: some changes are sharp and short, while some are slow and long-lasting (Figure 3). How do we effectively capture those multi-scale changes in the long geo-sensory time series?

To address this problem, we use *wavelet transform* to obtain a multi-resolution representation of the time series. As we will see, multi-scale changes appear clearly at different levels in the wavelet space and thus can be easily identified.

Wavelet transform. Wavelet transform [5] is a multi-resolution signal decomposition tool that provides time and frequency localization simultaneously. To decompose the geo-sensory time series, we choose the Haar wavelet transform due to its simplicity and practical effectiveness. Haar wavelet transform relies on a *scaling function* $\phi(t)$ and a *mother wavelet function* $\psi(t)$:

$$\phi(t) = \begin{cases} 1 & 0 \leq t < 1, \\ 0 & \text{otherwise.} \end{cases} \qquad \psi(t) = \begin{cases} 1 & 0 \leq t < 1/2, \\ -1 & 1/2 \leq t < 1, \\ 0 & \text{otherwise.} \end{cases}$$

Intuitively, given an input signal defined over the range $[0, 1]$, the scaling function $\phi(t)$ models the average strength of the signal, while the mother wavelet $\psi(t)$ models the detailed change of the signal. To decompose discrete time series data, the mother wavelet function $\psi(t)$ can be scaled and shifted to generate a set of orthogonal bases. Those bases capture changes that occur at different resolutions and locations. As a concrete example, Figure 4 shows the Haar wavelet bases for length-8 signals: (1) at level 1, $\psi_{1,1}$ models the change from $[t_1, t_4]$ to $[t_5, t_8]$; (2) at level 2, $\psi_{2,1}$ models the change from $[t_1, t_2]$ to $[t_3, t_4]$, and $\psi_{2,2}$ models the change from $[t_5, t_6]$ to $[t_7, t_8]$; and (3) at level 3, $\psi_{3,1}$ models the change from t_1 to t_2, $\psi_{3,2}$ models the change from t_3 to t_4, *etc*.

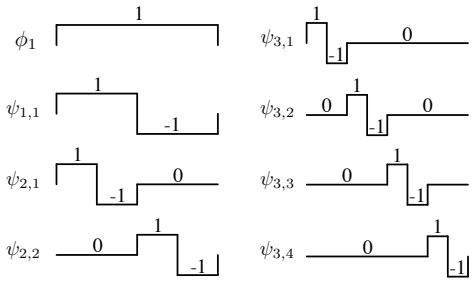


Figure 4: The Haar wavelet bases for length-8 signals.

With the Haar wavelet bases, an arbitrary time series s_i can be represented as a linear combination of them, namely

$$s_i = c_0 \cdot \phi_0 + \sum_{i=1}^{l} \sum_{j=1}^{k_i} c_{ij} \cdot \psi_{i,j},$$

where l is the total number of levels, k_i is the number of mother wavelets at level i, and c_{ij} is the wavelet coefficient for basis $\psi_{i,j}$. The wavelet coefficients can be efficiently obtained using pair-wise average-difference computation. Specifically, given a length-$2m$ signal $[v_1, v_2, \ldots, v_{2m-1}, v_{2m}]$, the pair-wise computation takes every two adjacent values $(v_{2j-1}, v_{2j})(1 \leq j \leq m)$ and compute

$$a_j = (v_{2j-1} + v_{2j})/2, \quad c_j = (v_{2j-1} - v_{2j})/2.$$

Here, c_j is returned as the wavelet coefficient for interval j at current level, and the average a_j is fed to the next higher level for further average-difference computation. For instance, Table 1 shows the process of computing the Haar wavelet coefficients for a length-8 signal $(1, 1, 6, 8, 9, 11, 15, 25)$.

Table 1: An illustration for wavelet coefficient computation.

Level	Average	Difference
	$(1, 1, 6, 8, 9, 11, 15, 25)$	
3	$(1, 7, 10, 20)$	$(0, -1, -1, -5)$
2	$(4, 15)$	$(-3, -5)$
1	(9.5)	(-5.5)

Extracting evolving intervals. In the wavelet representation of a signal, the coefficient c_{ij} measures the signal's strength of change in the j-th interval at level i: (1) a large positive c_{ij} means the signal decreases significantly in the interval; (2) a large negative c_{ij} means the signal increases significantly in the interval; and (3) a small absolute value of c_{ij} means the signal does not change much. The level i determines the resolution of observation; while the position j captures where the change occurs.

The above observation leads to our strategy for filtering trivial fluctuations and preserving multi-resolution changes in the geo-sensory time series. As shown in Algorithm 1, given a signal s_i, we obtain its wavelet coefficients and compare them with a change threshold δ. Specifically, we examine whether the absolute value of c_{ij} is larger than δ (line 5). If true, we add the corresponding timestamps into the results set T (line 6). The pre-specified threshold δ has a clear physical meaning as it measures how much change do we think is significant.

It is worth mentioning that, when extracting the evolving intervals, we only consider the levels larger than min_level. This is because the evolving behaviors of sensors are usually caused by external events (*e.g.*, air pollutions mostly result from heavy traffic, wind, manufacturing activity, *etc.*) and do not span a very long time period. Our goal is capture such short-term but frequently occurring changes, not the long-term trend in the geo-sensory data. Hence, we ignore the levels that are too coarse.

3.1.2 Mining Frequent Evolutions

Once the evolving intervals have been identified for each sensor s_i, next task is to detect evolving behaviors that frequently occur. Our observation is that, the evolution of a sensor may experience several different stages during each evolving interval. If we treat each evolving interval as a basic unit, we may not easily observe similar behaviors. However, if we break each interval into several segments, "similar" segments from different intervals can constitute frequent behaviors.

As an example, Figure 5 shows two evolving intervals, $\mathcal{I}_1$ and $\mathcal{I}_2$, of a sensor. $\mathcal{I}_1$ and $\mathcal{I}_2$ are clearly dissimilar if we treat each one as a whole. However, suppose we break $\mathcal{I}_1$ into five line segments and $\mathcal{I}_2$ into four, the clusters C_1, C_2, C_3, C_4 become evident. The line segments in each cluster are "similar" in the sense that they have very similar slopes.

Algorithm 1: Evolving interval extraction.

Input: Geo-sensory time series s_i, change threshold δ.
Output: The set of evolving timestamps in s_i.

```
1 C ← Haar wavelet coefficients of s_i;
2 T ← φ;
3 for i = min_level to l do
4     for j = 1 to k_i do
5         if |c_ij| ≥ δ then
6             Add into T the timestamps in interval j of level i;
7 return T;
```

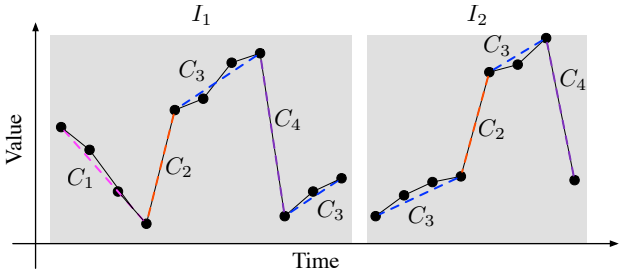


Figure 5: Segment-and-group of evolving intervals.

A segment-and-group approach. The above observation motivates us to first partition the evolving intervals into line segments, and then group similar line segments to detect frequent evolutions.

The segmentation of each evolving interval can be done with the widely adopted bottom-up approach (see details in [8]): for each evolving interval, we start from the smallest segments, and iteratively merge two neighboring segments into a larger one that has the minimum approximation error. The segmentation process terminates when every possible merge leads to an error larger than a threshold ϵ.

After segmenting each evolving interval, we obtain a set of result line segments that have different slopes. Our goal is to divide them into groups such that the segments in the same group have similar slopes. How do we do this without knowing the number of groups beforehand? Below, we design a grouping strategy based on mean shift [4], a non-parametric clustering method based on kernel density estimation. Compared with other classic clustering methods like K-means and DBSCAN, mean shift has several nice properties for our purpose. First, it does not assume any prior knowledge about the number of clusters or the data distribution. Thus it can effectively discover arbitrarily shaped clusters in a complex data space. Second, it has only one parameter, namely the bandwidth, which has a physical meaning as the scale of observation.

For a line segment l_i over the sub-interval $[t_s, t_e]$, we consider l_i as a data point x_i, where the value x_i is the slope of l_i (derived from the start timestamp t_s and the end timestamp t_e). Meanwhile, each x_i is associated with a weight w_i, which is the length of l_i, namely $t_e - t_s$. Based on mean shift, we cluster the data points by detecting modes (density maxima) and grouping the points that share the same mode. Specifically, for each data point, we find its mode by iteratively shifting a length-$2w$ window. The window is called the kernel window and the radius w is called the bandwidth. In each iteration, let $y^{(k)}$ be the center of the current window, and $\{x_1, x_2, \ldots, x_m\}$ be the m data points inside the win-

dow, then the window center is shifted to the weighted mean of $\{x_1, x_2, \ldots, x_m\}$, resulting in the maximum increase of density for $y^{(k)}$. The shifting operation leads to a new kernel window located at the weighted mean of $\{x_1, x_2, \ldots, x_m\}$, namely

$$y^{(k+1)} = \sum_{i=1}^{m} w_i x_i \Big/ \sum_{i=1}^{m} w_i. \quad (1)$$

As shown in Figure 6, for each point x, we start with an initial kernel window centered at $y^{(0)} = x$, and iteratively shift the window according to Equation 1. The sequence $\{y^{(k)}\}$ will converge to the mode that x belongs to. After performing the mean shift process for every point, we group the points that have the same mode into one cluster.

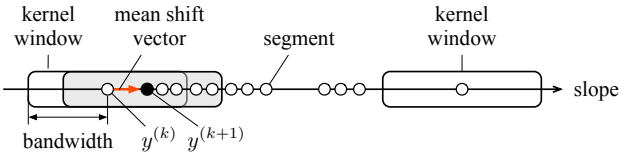


Figure 6: Cluster the segments using mean shift.

Frequent evolution discovery. Now we have obtained the clusters of similar segments, along with the matching evolving timestamps for each cluster. We consider each cluster as an evolution behavior, where the change rate range is derived from its matching timestamps, and the support is the number of the matching timestamps. Recall that we want to find *frequent* evolutions of the sensor. With the minimum support θ, we eliminate the clusters that have support lower than θ, then the remaining clusters are returned as frequent evolutions for the sensor.

3.2 Stage II: SCP Generation

For each sensor s, we now have a set $\mathcal{P}_s$ of its frequent evolutions, along with the matching timestamps for each pattern $p \in \mathcal{P}_s$. In the second stage of `Assembler`, we assemble the frequent evolutions of individual sensors into SCPs.

3.2.1 The Anti-monotonicity Property

THEOREM 1. *Let $G = \{s_1, s_2, \ldots, s_g\}$ be a set of spatially connected sensors, and $E_G = \{R_1, R_2, \ldots, R_g\}$ be an SCP on G. For any $G' = \{s_{i_1}, s_{i_2}, \ldots, s_{i_k}\} \subset G$ that is spatially connected, the co-evolution $E_{G'} = \{R_{i_1}, R_{i_1}, \ldots, R_{i_k}\}$ is an SCP on G'.*

PROOF. The proof is obvious because the matching timestamps of E_G must also match $E_{G'}$, hence the support of $E_{G'}$ must be no smaller than the minimum support θ. □

Theorem 1 amounts to saying that, if there are no SCPs over a set G' of sensors, then no supersets of G' can have any SCPs. This property ensures that, if we generate SCPs on large sensor sets by assembling the SCPs on small sensor sets, we effectively prune the search space without missing any patterns.

The pattern assembling operation is achieved via timestamp intersection. As shown in Figure 7, suppose we have extracted a frequent evolution P_1 for sensor s_1, and P_2 for sensor s_2. If s_1 and s_2 are spatially connected, then we merge P_1 and P_2 to generate a co-evolution over $\{s_1, s_2\}$, namely P_{12}. The matching timestamps of P_{12} is simply the common timestamps of P_1 and P_2. With the derived matching timestamps, it is trivial to compute the support of P_{12} and determine whether it is frequent or not. Note that Figure 7 is only an example for assembling one pair of patterns, when

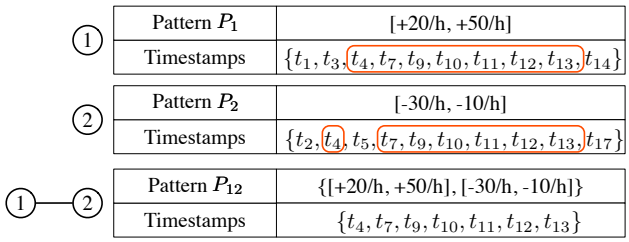


Figure 7: Find SCP by intersecting matching timestamps.

there are multiple patterns on both s_1 and s_2, we need to perform pairwise pattern assembling to obtain all the SCPs on $\{s_1, s_2\}$.

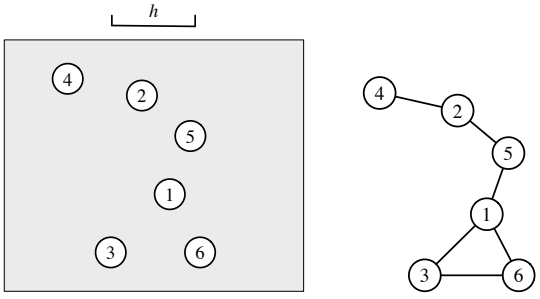
Based on the assembling operation and the anti-monotonicity property, one idea is to use the Apriori rule [1] to find SCPs in a bottom-up manner: starting with the single frequent evolutions, we examine pairs of spatially connected sensors to generate size-2 SCPs. Once the size-k SCPs have been discovered, we join every possible pair of size-k SCPs to generate size-$(k+1)$ candidate SCPs, and judge whether each candidate is frequent. Such a bottom-up process is terminated when no more SCPs exist for the current size.

Unfortunately, the Apriori-based mining process suffers from two problems: (1) to generate large-size SCPs, it needs to generate a huge number of small-size SCPs as candidates and keep them in memory, which incurs substantial overhead; and (2) when generating size-$(k+1)$ candidates from size-k patterns, it need to examine every pair of size-k patterns and determine whether they are joinable. It fails to take advantage of the spatial constraint and leads to a huge number of unnecessary comparisons.

3.2.2 The SCP Search Tree

We design a novel structure called the SCP search tree to facilitate more efficient SCP generation. We first introduce the concept of *connectivity graph*.

DEFINITION 8 (CONNECTIVITY GRAPH). *Given a set $\mathcal{S}$ of sensors and a distance threshold h, the connectivity graph $G_{\mathcal{S}}$ is constructed as follows: (1) each vertex in G corresponds to a sensor in $\mathcal{S}$; and (2) there is an edge between two vertices if their corresponding sensors have a distance no larger than h.*



(a) A set $\mathcal{S}$ of sensors and a distance threshold h.

(b) The connectivity graph $G_{\mathcal{S}}$.

Figure 8: The connectivity graph for the sensor set $\mathcal{S}$.

Figure 8 is an example of the connectivity graph. Recall that we want to find groups of spatially connected sensors to form SCPs. The connectivity graph well models the spatial constraint, because each set of spatially connected sensors in $\mathcal{S}$ uniquely corresponds to one connected component in the graph $G_{\mathcal{S}}$. Our problem is then reduced to finding all the connected components in $G_{\mathcal{S}}$ that have SCPs. However, since $\mathcal{S}$ can consist of hundreds of sensors, it is

prohibitively expensive to enumerate all the connected components in $G_{\mathcal{S}}$ and search for the SCPs. Below, we introduce the *neighbor* and *parent* relations between two connected components.

DEFINITION 9 (NEIGHBOR). *Given a connectivity graph G, let X be a size-k connected component in G, and Y a size-$(k+1)$ one. Y is a neighbor of X if Y includes all the members of X.*

DEFINITION 10 (PARENT). *Let Y be a size-$(k+1)$ connected component in a connectivity graph G. Given a vertex ordering $\mathcal{V}$, the roll-up operation on Y removes one vertex s from Y such that: (1) the result set $X = Y - \{s\}$ is still connected; (2) s is the first possible vertex in $\mathcal{V}$ on the premise of satisfying Condition (1). We say X is the parent of Y, and Y is a child of X.*

Clearly, any connected component Y has one unique parent. Hence, from any connected component Y, if we progressively perform the roll-up operation, we will reach the empty set ϕ.

EXAMPLE 2. *Consider the graph $G_{\mathcal{S}}$ in Figure 8(b). Suppose the vertex ordering is $\mathcal{V} = 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$. The roll-up operation on the connected component $\{245\}$ generates $\{25\}$, because sensor s_4 is the first possible one in $\mathcal{V}$ that ensures the remaining sensors are still connected. Similarly, the roll-up operation on $\{25\}$ generates its parent $\{5\}$. Finally, the roll-up operation on $\{5\}$ outputs ϕ.*

With the roll-up operation, all the connected components in the connectivity graph actually form a tree structure, with the empty set ϕ as the root. We call such a structure *the SCP search tree*. Each node in the tree stores a set of spatially connected sensors, as well as the SCPs occurring on them. Figure 9 show the SCP search tree for the sensors in Figure 8(a).

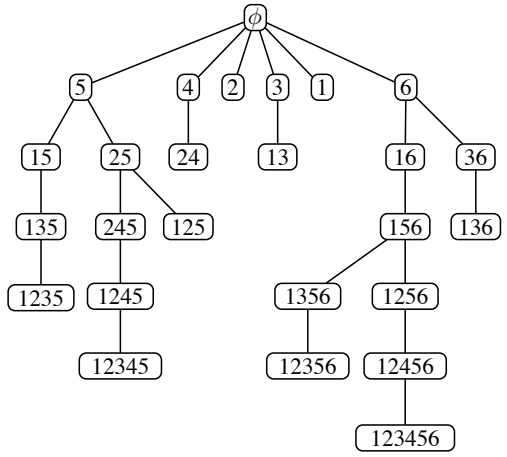


Figure 9: The SCP search tree.

The SCP search tree effectively organizes all the connected components into a tree structure based on the spatial constraint. The remaining concern is, how do we obtain the vertex ordering $\mathcal{V}$ in order to define the SCP search tree? Actually, $\mathcal{V}$ is only used to ensure the roll-up operation on any connected component generates a unique parent. Any choice of $\mathcal{V}$ can lead to a valid SCP search tree, and the cost of SCP generation is not sensitive to the specific choice of $\mathcal{V}$. In our implementation, we simply obtain $\mathcal{V}$ based on sensor id.

3.2.3 Depth-first SCP Search

Note that the SCP search tree organizes the connected components only *conceptually*. To generate SCPs, we do not need to construct the entire tree structure beforehand. Instead, we perform depth-first construction from the root node ϕ, and only visit the nodes that have SCPs. Specifically, for any node N in the tree, if N does not have any SCPs, then no descendants of N can have SCPs (Theorem 1), and the subtree rooted at N can be safely pruned.

Algorithm 2 sketches the depth-first SCP search process. To obtain all SCPs, we just need to feed Algorithm 2 with $X = \phi$, *i.e.*, searching from the root ϕ. As shown, given node X, we output the SCPs on X if X contains no less than one sensor. Then we start depth-first search from X. First, we find all the neighbors of X in the SCP search tree (line 3), which can be easily done by adding a new sensor that is adjacent to X in G. For each neighbor Y, we perform the roll-up operation to verify whether Y is indeed a child of X.[2] If true and Y contains SCPs (lines 5-7), we recursively perform depth-first search on Y (line 8), otherwise we safely prune all the subtree rooted at Y.

Algorithm 2: Search(G, X)

```
Input: Connectivity graph G, connected component X ⊆ G.
Output: The SCPs on X.
1 if |X| ≥ 1 then
2     Output the SCPs on X;
3 NS(X) ← neighbors of X in the SCP search tree;
4 foreach Y ∈ NS(X) do
5     if roll_up(Y) = X then
6         P_Y ← the SCPs on Y;
7         if P_Y is not empty then
8             Search(G, Y);
```

3.3 Discussions

Time Complexity. The cost of the first stage involves two parts: (1) wavelet transform, and (2) the segment-and-group process. For each sensor, the wavelet transform takes $O(m)$ time where m is the length of the time series. Now consider the segment-ang-group process. For each sensor, let n_e be the number of evolving intervals, l_e be the average length of the evolving intervals, and l_s the segment length, then the segmentation takes $O(n_e \cdot l_e \cdot l_s)$ time for each sensor. Since $n_e \cdot l_e < m$ and l_s is usually small, the segmentation time complexity is $O(m)$. For mean shift clustering, assume the number of segments is n_l and the average number of shifting operations is k, the time cost of mean shift is then $O(n_l \cdot k) \approx O(m)$. Hence, the total time complexity of the first stage is $O(nm)$.

For the second stage, let n_G be the number of connected components in G that have SCPs. During the depth-first search, Algorithm 2 are called n_G times. For each call of Algorithm 2, the time cost is determined by line 3, line 5, and line 6. Denote by $|E_G|$ the number of edges in G. For any connected component X in G, its neighbors can be obtained in $O(|E_G|)$ time (line 3). For line 5, each roll-up operation takes $O(|E_G|)$ time. For line 6, let n_p be the maximum number of SCPs on a connected component and n_s be the maximum support of an SCP. Since the SCPs of Y are derived by intersecting the timestamps of the SCPs on X and the SCPs on the sensor $Y - X$, the time cost of line 6 is $O(n_p^2 n_s)$. The `for` loop outside line 5 iterates at most n (the number of sensors) times as X can have at most n neighbors. Hence, the total time complexity of the second stage is $O(n_G(n|E_G| + n_p^2 n_s))$.

Space Complexity. The space complexity of the first stage is $O(m)$ as the space cost of both wavelet transform and the segment-and-group process is linear in input size. For the second stage, since

[2]Note that not every neighbor Y is necessarily a child of X.

`Assembler` performs depth-first search and examines one path at a time, there are at most n connected components maintained in memory. For each connected component, we need to maintain its SCPs along with the matching timestamps. Hence, the space complexity of the second stage is $O(n \cdot n_p \cdot n_s)$.

Parameter Setting. There are four parameters in `Assembler`: (1) the minimum support θ; (2) the distance threshold h; (3) the change threshold δ; and (4) the mean shift bandwidth ω. The first three paramters are easy to set based on application need, because it is intuitive to determine: (1) how many occurrences can be considered *frequent* enough; (2) what distance makes two sensors reachable *w.r.t.* the monitored condition; and (3) how much change in the reading reflects a significant and unusual behavior. The bandwidth parameter ω comes with the mean shift algorithm, and various techniques have been proposed to specify it. For example, the Scott's rule of thumb and the data-driven selection [3] are two popular approaches for this purpose.

4. EXPERIMENTS

In this section, we evaluate the empirical performance of the `Assembler` method. All the algorithms were implemented in JAVA and the experiments were conducted on a computer with Intel Core i7 2.4Ghz CPU and 8GB memory.

4.1 Experimental Setup

Data Sets. Our experiments are based on two real geo-sensory data sets and multiple synthetic data sets:

1. **Air** is an air quality data set. 180 air quality sensors are deployed in 16 cities in northern China (Beijing, Tianjin, and 14 cities in the Hebei Province). Each sensor has measured the hourly AQI during the period 2013.02.08 – 2014.08.27.

2. **Bike** is the Citi Bike rental data set[3]. For the 332 rental docks in New York, we record the number of available bikes at each dock every 30 minutes during 2013.07.01 – 2014.08.30.

3. **Syn-Sensor** is a collection of 4 synthetic data sets used to evaluate the scalability of `Assembler` *w.r.t.* the number of sensors n. We set $n = 100, 200, 400, 800$. For each n, we generate n uniformly distributed sensors. For each sensor, we first generate a length-100000 synthetic time series with the autoregressive model, then randomly insert 1000 length-10 changes into the time series.

4. **Syn-Length** is a collection of 4 synthetic data sets used to evaluate the scalability of `Assembler` *w.r.t.* the length of time series m. We set $m = 10^3, 10^4, 10^5, 10^6$. For each m, we first generate 200 randomly distributed sensors. For each sensor, we mix a length-m synthetic autoregressive time series with $0.01m$ length-10 changes.

Compared Method. To the best of our knowledge, no existing methods can be directly used for SCP mining in geo-sensory data. In order to compare with `Assembler`, we design a baseline method called `WaveApriori`. It is also a two-stage method and shares the same first stage with `Assembler`. However, in the second stage, `WaveApriori` uses the Apriori rule for SCP search (Section 3.2.1) instead of the SCP search tree.

[3]https://www.citibikenyc.com/

4.2 Experimental Results

In this subsection, we report our experimental results. Below, we first examine several illustrating SCPs discovered by `Assembler` on the two real data sets. Then, we study the efficiency and scalability of `Assembler` under various parameter settings.

4.2.1 Illustrating Patterns

For SCP mining, four parameters need to be specified: (1) the minimum support θ, (2) the connectivity distance h, (3) the change threshold δ, and (4) the mean shift bandwidth w. On **Air**, we set $\theta = 200, h = 30$ km, $\delta = 30, w = 10$. Note that the AQI change of 30 ($\delta = 30$) indicates quite evident increase/decrease of air quality. Under such a setting, `Assembler` obtains 8328 SCPs in total, we are particularly interested in the SCPs that involve a large number of sensors. Hence, we demonstrate in Figure 10 two size-10 patterns. We can see that the first pattern P_1 is a more localized pattern that occur on the sensors in Beijing. In P_1, the sensors s_1, s_2, s_3 in the periphery of Beijing have evident AQI decrease, while the sensors $s_4, \ldots, s_{10}$ in downtown Beijing shows evident AQI increase. We have also investigated the time and the wind direction when P_1 occurs. Interestingly, the occurring hours of P_1 show clear peaks in two intervals: [6am-9am] and [8pm-10pm]. During these hours, the traffic may move from the periphery area to the downtown area for work and nightlife activities, thus deteriorating the air quality in downtown. For the pattern P_2, the AQIs of the sensors $s_1, \ldots, s_4$ decrease while the AQIs of $s_5, \ldots s_{10}$ increase. As shown in Figure 10, the occurring hours of this pattern seem to be randomly distributed, but the wind direction is mostly West when this pattern occurs. It is quite likely the wind carries the air pollutants from $s_1, \ldots, s_4$ to $s_5, \ldots s_{10}$ and causes this pattern.

Figure 11 shows two example patterns on the **Bike** data set, with the parameters $\theta = 150, h = 1$ km, $\delta = 10, w = 3$. Both patterns occur mostly in the morning. For P_1, the number of bikes decreases for the docks $(s_4, \ldots, s_7)$ around the ferry station; while the number of bikes increases for the docks (s_1, s_2, s_3) in the office zone. This pattern is probably because people arrive at Manhattan by ferry in the morning, and then rent bikes to ride to their offices. The situation is similar for the pattern P_2, the docks $s_5, \ldots, s_9$ are located in a residence area, while the docks $s_1, \ldots, s_4$ are in downtown Manhattan. Many people may choose to rent bikes to go to work due to the heavy traffic in the morning.

4.2.2 Efficiency Study

In this subsection, we compare the efficiency of `Assembler` and `WaveApriori`. Since `Assembler` works in two stages, we break its cost into two parts: Assembler-1 for the first stage, and Assembler-2 for the second. `WaveApriori` is also a two-stage method and has the same first stage as `Assembler`, we use Apriori to denote the cost of its second stage.

As aforementioned, there are four parameters for SCP mining: θ, h, δ, and w. In the following, we study the effect of each parameter while the other parameters are fixed at their default values. Table 2 shows our parameter settings on the **Air** and **Bike** data sets, where the numbers in bold denote the default values.

Table 2: Parameter settings on the Air and Bike data sets.

Parameter	Air	Bike
θ	150, **200**, 250, 300	100, **150**, 200, 250, 300
h	5, 10, 15, 20, 25, **30**	0.5, 1.0, 1.5, **2.0**, 2.5
δ	15, 20, 25, **30**	6, 8, **10**, 12
w	6, 8, **10**, 12	2, **3**, 4, 5

Varying θ. In the first set of experiments, we examine the effect of the minimum support θ on the performance of `Assembler`

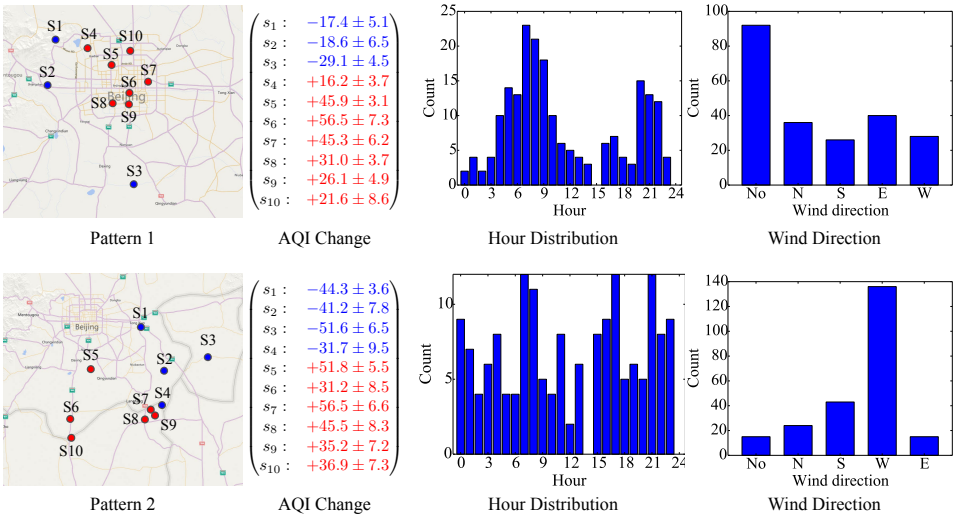


Figure 10: Two example patterns on the Air data set.

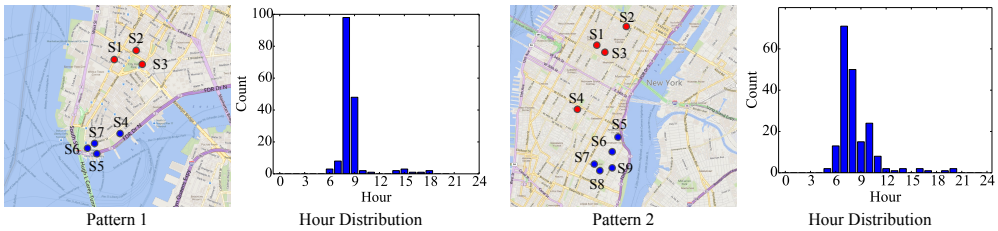
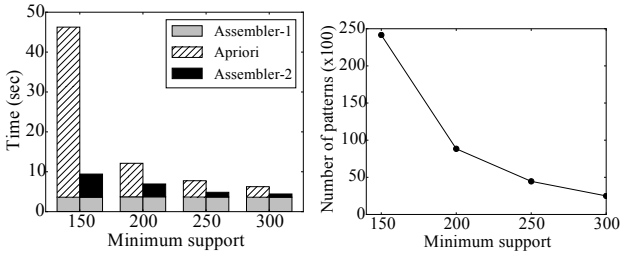


Figure 11: Two example patterns on the Bike data set.

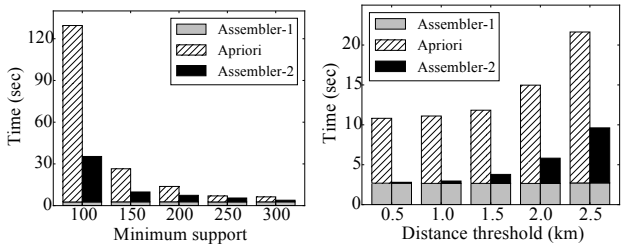
and `WaveApriori`. As shown in Figure 12(a), the running time of both methods decreases with θ on the **Air** data set. The decrease comes from the second stage for both methods. With larger θ, fewer sets of spatially connected sensors can have SCPs and the pattern search space is smaller for both methods. This phenomenon can be observed in Figure 12(b), where the number of SCPs decreases rapidly with θ on the **Air** data set. Comparing the running time of the two methods, we can see the second stage of `Assembler` is much faster than that of `WaveApriori`. This is because `Assembler` effectively leverages the spatial constraint when generating SCPs from single frequent evolutions. Figure 13(a) shows the running time of the two methods on the **Bike** data set, where similar trends are observed.



(a) Running time. (b) Number of patterns.

Figure 12: Varying minimum support θ on the Air data set.

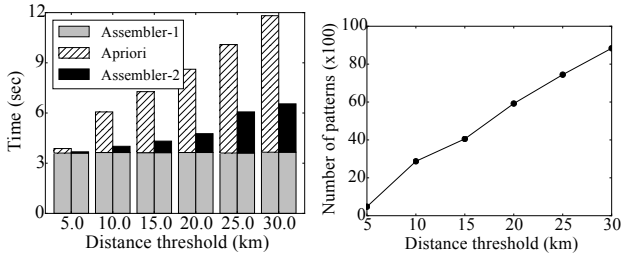
Varying h. In the second set of experiments, we evaluate the performance of `Assembler` and `WaveApriori` as the distance threshold h varies. Figure 14(a) and 13(b) show the running time of the two methods on the **Air** and **Bike** data sets, respectively. The first stage of the two methods is not affected by h, while the cost of the second stage increases with h for both methods. This is expected. As h increases, the number of spatially connected sensor set becomes larger, leading to more SCPs and a larger pattern



(a) Time *v.s.* θ. (b) Time *v.s.* h.

Figure 13: Running time comparison on the Bike data set.

search space. As shown in Figure 14(b), the number of SCPs increases roughly linearly with h on the **Air** data set. Meanwhile, one can observe again the second stage of `Assembler` is much more efficient than that of `WaveApriori`. The performance gap is particularly significant when h is very small. This phenomenon suggests that the SCP search tree indeed effectively utilizes the distance constraint to largely prune the search space.



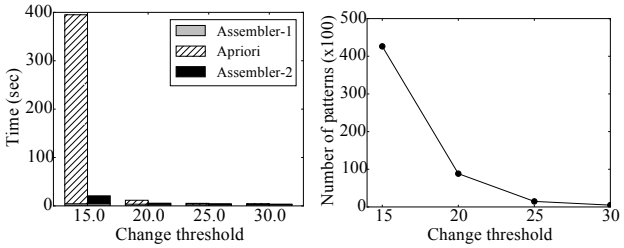
(a) Running time. (b) Number of patterns.

Figure 14: Varying distance threshold h on the Air data set.

In practice, h should be determined with the guidance of domain knowledge. On the **Air** data set, we set h to 5km - 30km because

air pollution can be dispersed faraway due to factors such as wind and traffic. In contrast, on the **Bike** data set, we set h to 0.5km - 3km, as people usually take short trips by bike.

Varying δ. We proceed to study the effect of the change threshold δ. Figure 15(a) and 15(b) give the results on the **Air** data set. When extracting evolving intervals using wavelet transform, we set min_level to $l-3$, namely we only concern about the three most detailed levels. From Figure 15(a), we see that, the running time of both methods becomes larger when δ is smaller, and this trend is especially obvious for `WaveApriori`. Recall that δ specifies how much change we think is significant. When δ is small, many small evolutions are also extracted, and thus the number of result SCPs becomes much larger (Figure 15(b)).

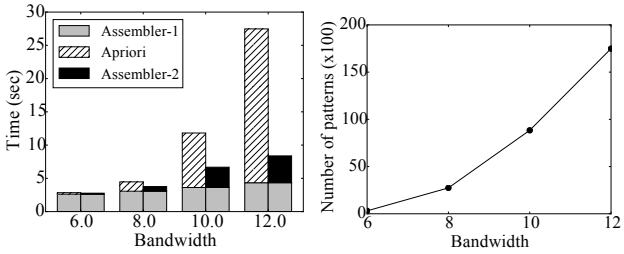


(a) Running time. (b) Number of patterns.

Figure 15: Varying change threshold δ on the Air data set.

The performance gap between the two methods is extremely large when δ is small. This is because `WaveApriori` generates numerous small candidate patterns for a small δ, the pair-wise candidate comparison incurs substantial computation overhead, deteriorating its performance rapidly. Similar results are observed on the **Bike** data set, we omit them due to the space limit.

Varying w. Finally, we study the performance of the two methods when the mean shift bandwidth w varies. Note that w controls the clustering granularity, when w is large, more data points are grouped together and thus the co-evolutions on spatially connected sensors have better chances to exceed the minimum support θ. Figure 16(a) and 16(b) show that the running time and the number of SCPs increase for both methods. Specifically, the running time of the first stage increases slightly with w, while the cost of the second stages grows more rapidly.

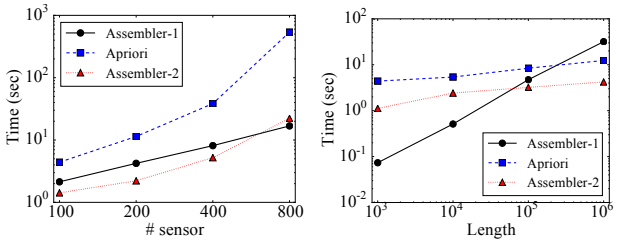


(a) Running time. (b) Number of patterns.

Figure 16: Varying bandwidth w on the Air data set.

4.2.3 Scalability

In this subsection, we study the scalability of `Assembler` and `WaveApriori` using the two collections of synthetic data sets. The four data sets in **Syn-sensor** have the same time series length m but different numbers of sensors n. Figure 17(a) shows the running time of `Assembler` and `WaveApriori` as n increases, with $\theta = 1000$ and h set to the average distance between the sensors. As shown, the cost of the first stage increases linearly with n for both methods, which is expected according to our theoretical analysis in Section 3.3. For the second stage, the cost of both methods increases super-linearly with n. This is due to the combinatorial nature of SCP: when n increases, the number of spatially connected sensor set increases rapidly, resulting in larger cost for SCP search. That said, we notice that the running time of `Assembler` increases more slowly than `WaveApriori`. It is because `Assembler` more effectively leverages the spatial constraint to prune the search space.



(a) Time *v.s.* number of sensors n. (b) Time *v.s.* sequence length m.

Figure 17: Scalability study.

For the other collection **Syn-length**, the four data sets have the same number of sensors n, but different time series length m. Figure 17(b) shows the running time of the two methods as m varies, with $\theta = 0.01 \cdot m$ and h set to the average distance of the sensors. As shown, the first stage of both methods still increases linearly with m, while the second stages increases sub-linearly. This suggests that `Assembler` scales well with the length of time series and suitable for massive geo-sensory data in practice.

5. RELATED WORK

Generally, our work is related to the following topics.

Motif discovery. Motif discovery in time series has been studied extensively. Given a distance threshold δ and a length l, two length-l time series subsequences form a motif if their distance is smaller than δ. The top-K motif discovery problem aims at finding the K subsequences that have the largest number of matches in the time series. Lin *et al.* [10] first introduced this problem and used hash-based counting to find motifs. Chiu *et al.* [2] developed an algorithm that finds approximate motifs in linear time. The key idea is to maintain a collision matrix and use random projection to map subsequences into the collision matrix. Mueen *et al.* [15] proposed a method that fast finds exact motifs with the linear ordering heuristic and the early abandoning strategy.

Motif discovery in multi-dimensional time series has also been studied. Tanaka *et al.* [20] first transformed the multi-dimensional time series using principal component analysis, and then used random projection [2] to find motifs in the transformed signal. Minnen *et al.* [14] studied the problem of mining sub-dimensional motifs that span across only a subset of the dimensions. Patel *et al.* [17] introduced lag patterns to capture the invariant ordering of the motifs extracted from multiple time series. Unfortunately, as discussed in Section 1, none of these techniques are capable of mining SCPs because of the redundancy problem of geo-sensory data, as well as the duration variation and combinatorial nature of SCP.

Time series segmentation and change detection. Sliding window, top-down, and bottom-up approaches [8] are popular methods to partition a time series into line segments. Wang *et al.* [22] proposed the pattern-based hidden Markov model that can segment a time series as well as learn the relationships between segments. Methods have also been proposed [9, 6] to obtain piecewise polynomial approximations and/or perform on-line segmentation.

Change detection aims to find the time points where the statistical property of the time series changes significantly. It is closely

related to time series segmentation as such points can be considered as the boundaries of different segments. Yamanishi *et al.* [23] unified the problems of change detection and outlier detection based on the on-line learning of an autoregressive model. Sharifzadeh *et al.* [19] used wavelet footprints to find the points where the polynomial curve fitting coefficients show discontinuities. Kawahara *et al.* [7] judged whether a point is a change by computing the probability density ratio of the reference and test intervals.

While `Assembler` uses the bottom-up segmentation approach due to its simplicity and practical effectiveness, it can be easily adapted to other segmentation algorithms. It is also worth mentioning that, the segmentation of `Assembler` is performed on short evolving intervals instead of the original long time series, which renders the segmentation process really fast.

Co-evolving time series analysis. Papadimitriou *et al.* [16] introduced SPIRIT for multi-sensory time series, with a focus on discovering the hidden variables and summarizing the key trends for the entire time series collection. Sakurai *et al.* [18] studied the problem of detecting all pairs of time series that have strong lag correlations. Ma *et al.* [11] designed a system for analyzing the sensory data in water distribution systems. One component in the system evaluates the correlations among the co-evolving multi-sensory data. However, it simply computes the Pearson's correlations between two sensors for a specific time interval, and then clusters the sensors that have similar trends. In contrast, our SCP does not rely on a specified time interval, and the member sensors in a SCP do not necessarily have similar trends.

Trasarti *et al.* [21] studied the problem of finding regions that show similar deviations in population density using mobile phone data. They assume the condition has periodicity, *i.e.*, the daily population densities in a region are similar in different days. While this assumption is reasonable for population density, it does not hold in many geo-sensory applications like air quality monitoring. Moreover, they extract vertical changes in population density by comparing the same hour of different days. In contrast, we extract the horizontal changes, *i.e.*, comparing the condition in current time interval with the previous time interval.

Matsubara *et al.* [13] proposed a model to summarize the global-level (*i.e.*, country-level) and the local-level (*i.e.*, state-level) properties of different diseases. However, it is designed specifically for epidemic data instead of general geo-sensory data. Matsubara *et al.* [12] also designed the AutoPlait model. However, the goal is to automatically discover the latent states for a bundle of time series and segment the time series, which clearly differs from our problem.

6. DISCUSSIONS AND CONCLUSIONS

We introduced the problem of mining spatial co-evolving patterns from geo-sensory data. We designed the two-stage method, `Assembler`, for effective and efficient SCP mining. It first detects frequent evolutions for individual sensors and then assembles single patterns into SCPs. Our experimental results demonstrate that `Assembler` is effective, efficient, and scalable. We note that `Assembler` also enjoys the nice property that both its two stages can be easily parallelized to achieve even better efficiency. For the first stage, the wavelet transform and the segment-and-group process are performed independently for the sensors. For the second stage, the branches from the single sensors in the SCP search tree are mutually independent, and thus can be searched in parallel. For future work, we plan to leverage the discovered SCPs to improve the air quality prediction task in different cities in China[4], and also perform a thorough study on the correlation between AQI SCPs and the traffic patterns in Beijing.

[4]http://urbanair.msra.cn/

7. ACKNOWLEDGEMENTS

We thank the reviewers for their insightful comments. This work was sponsored in part by the U.S. Army Research Lab. under Cooperative Agreement No. W911NF-09-2-0053 (NSCTA), National Science Foundation IIS-1017362, IIS-1320617, and IIS-1354329, HDTRA1-10-1-0120, and grant 1U54GM114838 awarded by NIGMS through funds provided by the trans-NIH Big Data to Knowledge (BD2K) initiative (www.bd2k.nih.gov), and MIAS, a DHS-IDS Center for Multimodal Information Access and Synthesis at UIUC. Xiuli Ma is supported by the National Natural Science Foundation of China under Grant No.61103025 and China Scholarship Council.

8. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, pages 487–499, 1994.

[2] B. Y. Chiu, E. J. Keogh, and S. Lonardi. Probabilistic discovery of time series motifs. In *KDD*, pages 493–498, 2003.

[3] D. Comaniciu. An algorithm for data-driven bandwidth selection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(2):281–288, 2003.

[4] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(5):603–619, 2002.

[5] I. Daubechies. The wavelet transform, time-frequency localization and signal analysis. *IEEE Transactions on Information Theory*, 36(5):961–1005, 1990.

[6] E. Fuchs, T. Gruber, J. Nitschke, and B. Sick. Online segmentation of time series based on polynomial least-squares approximations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(12):2232–2245, 2010.

[7] Y. Kawahara and M. Sugiyama. Change-point detection in time-series data by direct density-ratio estimation. In *SDM*, pages 389–400, 2009.

[8] E. J. Keogh, S. Chu, D. Hart, and M. Pazzani. Segmenting Time Series: A Survey and Novel Approach. In *Data Mining In Time Series Databases*, volume 57, pages 1–22. 2004.

[9] E. J. Keogh, S. Chu, D. M. Hart, and M. J. Pazzani. An online algorithm for segmenting time series. In *ICDM*, pages 289–296, 2001.

[10] J. Lin, E. Keogh, S. Lonardi, and P. Patel. Finding motifs in time series. In *Proceedings of the Second Workshop on Temporal Data Mining*, pages 53–68, 2002.

[11] X. Ma, H. Xiao, S. Xie, Q. Li, Q. Luo, and C. Tian. Continuous, online monitoring and analysis in large water distribution networks. In *ICDE*, pages 1332–1335, 2011.

[12] Y. Matsubara, Y. Sakurai, and C. Faloutsos. Autoplait: automatic mining of co-evolving time sequences. In *SIGMOD*, pages 193–204, 2014.

[13] Y. Matsubara, Y. Sakurai, W. G. van Panhuis, and C. Faloutsos. FUNNEL: automatic mining of spatially coevolving epidemics. In *KDD*, pages 105–114, 2014.

[14] D. Minnen, C. L. Isbell, I. A. Essa, and T. Starner. Detecting subdimensional motifs: An efficient algorithm for generalized multivariate pattern discovery. In *ICDM*, pages 601–606, 2007.

[15] A. Mueen, E. J. Keogh, Q. Zhu, S. Cash, and M. B. Westover. Exact discovery of time series motifs. In *SDM*, pages 473–484, 2009.

[16] S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming pattern discovery in multiple time-series. In *VLDB*, pages 697–708, 2005.

[17] D. Patel, W. Hsu, M. Lee, and S. Parthasarathy. Lag patterns in time series databases. In *DEXA*, pages 209–224, 2010.

[18] Y. Sakurai, S. Papadimitriou, and C. Faloutsos. BRAID: stream mining through group lag correlations. In *SIGMOD*, pages 599–610, 2005.

[19] M. Sharifzadeh, F. Azmoodeh, and C. Shahabi. Change detection in time series data using wavelet footprints. In *SSTD*, pages 127–144, 2005.

[20] Y. Tanaka, K. Iwamoto, and K. Uehara. Discovery of time-series motif from multi-dimensional data based on MDL principle. *Machine Learning*, 58(2-3):269–300, 2005.

[21] R. Trasarti, A.-M. Olteanu-Raimond, M. Nanni, T. Couronné, B. Furletti, F. Giannotti, Z. Smoreda, and C. Ziemlicki. Discovering urban and country dynamics from mobile phone data with spatial correlation patterns. *Telecommunications Policy*, 2014.

[22] P. Wang, H. Wang, and W. Wang. Finding semantics in time series. In *SIGMOD*, pages 385–396, 2011.

[23] K. Yamanishi and J. Takeuchi. A unifying framework for detecting outliers and change points from non-stationary time series data. In *KDD*, pages 676–681, 2002.

[24] Y. Zheng, L. Capra, O. Wolfson, and H. Yang. Urban computing: Concepts, methodologies, and applications. *ACM TIST*, 5(3):38:1–38:55, 2014.