Local Pair and Bundle Discovery over Co-Evolving Time Series

Georgios Chatzigeorgakidis
Dept. of Inf. & Telecommunications
University of Peloponnese, Greece
chgeorgakidis@uop.gr

Dimitrios Skoutas
IMSI, Athena R.C., Greece
dskoutas@imis.athena-innovation.gr

Kostas Patroumpas
IMSI, Athena R.C., Greece
kpatro@imis.athena-innovation.gr

Themis Palpanas
LIPADE, Paris Descartes University, France
themis@mi.parisdescartes.fr

Spiros Athanasiou
IMSI, Athena R.C., Greece
spathan@imis.athena-innovation.gr

Spiros Skiadopoulos
Dept. of Inf. & Telecommunications
University of Peloponnese, Greece
spiros@uop.gr

ABSTRACT

Time series exploration and mining has many applications across several industrial and scientific domains. In this paper, we consider the problem of detecting locally similar pairs and groups, called bundles, over co-evolving time series. These are pairs or groups of subsequences whose values do not differ by more than ϵ for at least δ consecutive timestamps, thus indicating common local patterns and trends. We first present a baseline algorithm that performs a sweep line scan across all timestamps to identify matches. Then, we propose a filter-verification technique that only examines candidate matches at judiciously chosen checkpoints across time. Specifically, we introduce two block scanning algorithms for discovering local pairs and bundles respectively, which leverage the potential of checkpoints to aggressively prune the search space. We experimentally evaluate our methods against real-world and synthetic datasets, demonstrating a speed-up in execution time by an order of magnitude over the baseline.

CCS CONCEPTS

• **Information systems → Spatial-temporal systems**.

KEYWORDS

time series, pair discovery, bundle discovery, local similarity

1 INTRODUCTION

Time series are important in many applications both in industry, e.g. in energy and finance, as well as in science, e.g. in astronomy and biology [16]. Therefore, efficient management and mining of time series is a task of critical importance, but also highly challenging due to the large volume and complex nature of this data.

Co-evolving time series are time series that are time-aligned, i.e. they contain observation values at the same timestamps all along their duration. In this work, we focus on discovering all *pairs* or *groups* (called *bundles*) of *locally* similar co-evolving time series and extracting the *subsequences* where this local similarity occurs. We consider two time series as *locally similar* if the pairwise distance of their values per timestamp is at most ϵ for a time interval that lasts at least δ *consecutive* timestamps.

Several research efforts have focused on similarity search over time series to detect patterns within a single or across a set of time series [14, 17, 22, 27]. However, to the best of our knowledge, the problem of discovering pairs or groups of similar *time-aligned subsequences* within a set of *co-evolving* time series has been overlooked.

Discovering such pairs and bundles is useful in various applications. For instance, public utility companies employ smart meters to collect time series measuring consumption per household (e.g., for water or electricity). Identifying such bundles of time series (i.e., a number of similar subsequences over certain time intervals) can reveal similar patterns of consumption among users, allowing for more personalized billing schemes. In finance, examining time series of stock prices can identify pairs or bundles of stocks trending similarly at competitive prices over some trading period, hence offering precious insight for possible future investments.

Figure 1 illustrates an example comprising four time series depicted with different colors. We observe that from timestamp 1 to 5 the values of T_1 and T_2 are very close to each other, thus forming a locally similar pair. Similarly, from timestamp 8 to 12, the values of T_1, T_2 and T_4 are close to each other, forming a bundle with three members. Note that values in each qualifying subsequence may fluctuate along a bundle as long as they remain close to the respective values per timestamp of the other members in that bundle.
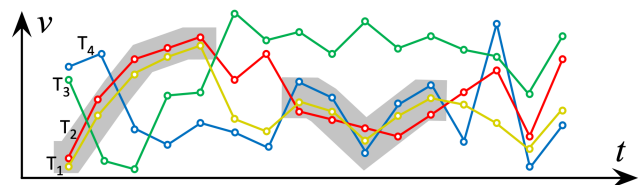
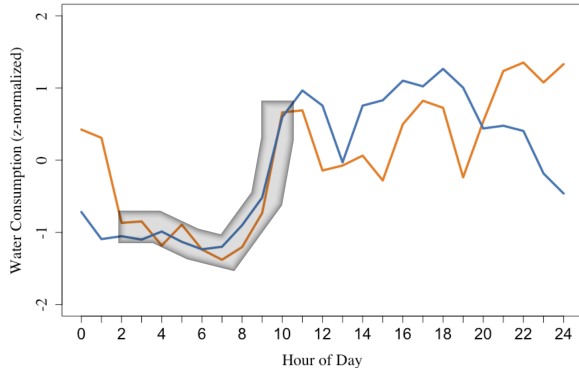**Figure 1: A pair and a bundle of locally similar time series.**

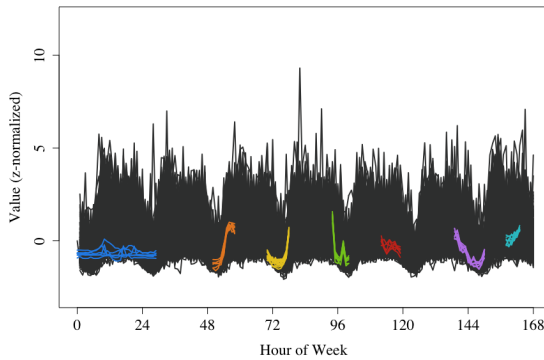Figure 2: Pair of locally (but not globally) similar time series.

Figure 3: Bundles of locally similar time series.

A real-world example is depicted in Figure 2. These two time series represent per-hour average water consumption during a day of the week for two different households. We can observe that their respective values per timestamp (at granularity of hours, in this example) are very close to each other during a certain time period (hours 2-11), but are farther apart in the rest. Hence, an algorithm that measures the global similarity between two time series might not consider this pair as similar; however, the subsequences inside the gray strip are clearly pairwise similar, and might indicate an interesting pattern. Identifying such local similarities within a sufficiently long time interval is our focus in this paper.

Furthermore, Figure 3 depicts several bundles of locally similar time series detected by our algorithms in a real-world dataset containing smart water meter measurements. The detected bundles represent different per-hour average water consumption patterns during a week. There is a wider pattern detected among 6 households during the first 30 hours of the week indicating reduced consumption (probably no permanent residence). The orange and yellow patterns indicate different morning routines during the third and fourth day of the week. The green and purple patterns represent a reduction in consumption during the late hours of the fourth and sixth day, respectively, with some intermediate consumption taking place during the night. Finally, the shorter red and light blue bundles suggest different evening patterns for two other days (respectively, decreasing and increasing consumption).

Discovering all possible pairs and bundles of locally similar time series, along with the corresponding subsequences, within large

sets is a computationally expensive process. To find matches, a filter-verification technique can be applied. At each timestamp, the *filtering* step can discover candidate pairs or groups having values close to each other; then, the *verification* step is invoked to determine whether each such candidate satisfies the required conditions, essentially whether this match occurs throughout a sufficiently large time interval. However, both the filtering and the verification steps are expensive. The computational cost becomes especially high for the case of bundle discovery, as it has to examine all possible subsets of locally similar time series that could form a bundle. Hence, such an exhaustive search is prohibitive when the number and/or the length of the time series is large.

In this paper, we employ a value discretization approach that divides the value axis in ranges equal to the value difference threshold ϵ, in order to reduce the number of candidate pairs or bundles that need to be checked per timestamp. Leveraging this, we first propose two *sweep line* scan algorithms, for pair and bundle discovery respectively, which operate according to the aforementioned filter-verification strategy. However, this process still incurs an excessive amount of comparisons, as it needs to scan all values at every timestamp. To overcome this, we introduce a more aggressive filtering that only checks at selected *checkpoints* across time, but ensuring that no false negatives ever occur. This approach incurs significant savings in computation cost, as we only need to examine candidate matches on those checkpoints only instead of all timestamps. To further reduce the number of examined candidates, we propose a strategy that judiciously places these checkpoints across the time axis in a more efficient manner. We then exploit these optimizations introducing two more efficient algorithms that significantly reduce the execution cost for both pair and bundle discovery.

The bundle discovery problem we address in this paper resembles the problem of *flock discovery in moving objects*, where the goal is to identify sufficiently large groups of objects that move close to each other over a sufficiently long period of time [1, 10, 24, 25]. In fact, the baseline algorithm we describe can be viewed as an adaptation of the algorithm presented in [25]. However, to the best of our knowledge, ours is the first work to address the problems of locally similar pair and bundle discovery over co-evolving time series. Our main contributions can be summarized as follows:

- We introduce the problems of local pair and bundle discovery over co-evolving time series.
- We suggest an aggressive checkpoint-based pruning method that drastically reduces the candidate pairs and bundles that need to be verified, significantly improving performance.
- We conduct an extensive experimental evaluation using both real-world and synthetic time series, showing that our algorithms outperform the respective sweep line baselines.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 describes the problems. Sections 4 and 5 introduce our algorithms for pair and bundle discovery, respectively. Section 6 reports our experimental results, and finally Section 7 concludes the paper.

2 RELATED WORK

Time series similarity. Similarity search over time series has attracted a lot of research interest [8]. One well-studied family of

approaches includes wavelet-based methods [5], which rely on *Discrete Wavelet Transform* [9] to reduce the dimensionality of time series and generate an index using the coefficients of the transformed sequences. The *Symbolic Aggregate Approximation* (SAX) representation [13] has led to the design of a series of indices, including *i*SAX [23], *i*SAX 2.0 [3], *i*SAX2+ [4], ADS+ [29], Coconut [12], DPiSAX [26], and ParIS [21]. However, these indices support similarity search over complete time series, i.e. whole-matching. Recently, the *ULISSE* index was proposed [14], which is the first index that can answer similarity search queries of variable length.

Moreover, many approaches have been proposed for *subsequence matching*. In this problem, a query subsequence is provided and the goal is to identify matches of it across one or more time series, typically of large length. The *UCR suite* [22] offers a framework comprising four different optimizations regarding subsequence similarity search. In computing *full-similarity-joins* over large collections of time series, i.e., to detect for each possible subsequence its *nearest neighbor*, the *matrix profile* [27] keeps track of Euclidean distances among each pair within a *similarity join set* (i.e., a set containing pairs of each subsequence with its nearest neighbor).

The problem we address in this paper differs from the above settings. Instead of identifying matches of a query subsequence against one, or more time series, we are interested in discovering locally similar pairs and bundles of time-aligned subsequences within a given collection of time series.

Correlated time series. Identifying similar subsequences between time series also indicates some *correlation* between them. Several approaches compute pairwise statistics (e.g., Pearson correlation, beta values) especially in streaming time series [6, 18, 28]. There are also works concerning *co-evolving* time series data, either towards detecting and correcting missing values [2] or mining typical patterns and points of variation to achieve a meaningful segmentation of large time series [15]. However, none of these approaches is applicable to our setting, where we require similarity in the time series values.

Time series clustering. Our work also relates to *clustering of time series*, where methods perform either partitioning or density-based clustering. In the former class, algorithms typically partition the time series into k clusters. Similarly to iterative refinement employed in k-means, the k-Shape partitioning algorithm [19, 20] aims to preserve the shapes of time series assigned to each cluster by considering the shape-based distance, a normalized version of the cross-correlation measure between time series. In contrast, density-based clustering methods are able to identify clusters of time series with arbitrary shapes. YADING [7] is a highly efficient and accurate such algorithm, which consists of three steps: it first samples the input time series also employing PAA (Piecewise Aggregate Approximation) to reduce the dimensionality, then applies multi-density clustering over the samples, and finally assigns the rest of the input to the identified clusters. However, clustering methods consider time series in their entirety and not matching subsequences as we consider in this work.

Discovery of movement patterns in trajectories. Our work also relates to approaches for discovering clusters of moving objects, in particular a type of movement patterns that is referred to as *flocks* [10]. A flock is a group of at least m objects moving together within a circular disk of diameter ϵ for at least δ consecutive

timestamps. Finding an exact flock is NP-hard, hence this work suggests an *approximate* solution to find the *maximal* flock from a set of trajectories using computational geometry concepts. In [1], another *approximate* solution for detecting all flocks is based on a skip-quadtree that indexes sub-trajectories. Flock discovery over *streaming* positions from moving objects was addressed in [25]. This *exact* solution discovers flock disks that cover a set of points at each timestamp. Their flock discovery algorithm finds candidate flocks per timestamp and joins them with the candidate ones from the previous timestamps, reporting a flock as a result when it exceeds the time constraint δ. An improvement over this technique was presented in [24], using a *plane sweeping* technique to accelerate detection of object candidates per flock at each timestamp, while an inverted index speeds up comparisons between candidate disks across time. In our setting, detection of bundles is similar to flocks, thus for our baseline method we adapt the algorithm from [25].

3 PROBLEM DEFINITION

A *time series* is a time-ordered sequence $X = \{X^1, X^2 \dots, X^k\}$, where X^i is the value at the i-th timestamp and k is the length of the series (i.e., the number of timestamps). We consider a set of *co-evolving* time series, so all time series are *time-aligned* and each series has a value at each of the k timestamps. Given a set of such co-evolving time series, our goal is to find *pairs* of time series that have similar values locally over some time intervals of significant duration. More specifically:

DEFINITION 1 (LOCALLY SIMILAR TIME SERIES). *Two co-evolving time series X_i and X_j are locally similar if there exists a time interval I spanning at least δ consecutive timestamps such that at every timestamp in I their corresponding values do not differ by more than a given threshold ϵ, i.e., $\forall t \in I, |X_i^t - X_j^t| \le \epsilon$.*

Note that threshold ϵ expresses the maximum tolerable deviation per timestamp between two time series, so it actually concerns the absolute difference of their corresponding values. We wish to find all such pairs of time series, so the problem is actually a *self-join* over the dataset, specifying as join criteria the distance threshold ϵ and the minimum time duration δ of qualifying pairs. More formally:

PROBLEM 1 (PAIR DISCOVERY OVER TIME SERIES). *Given a set of n co-evolving time series $X = \{X_1, X_2, ..., X_n\}$ of equal duration k, a distance threshold $\epsilon > 0$, and a time duration threshold $\delta > 1$ timestamps, $(\delta \in N)$, retrieve all pairs $\{X_i, X_j\}, 1 \le i < j \le n$ of locally similar time series along with the corresponding time intervals.*

For example, in Figure 4, the detected pairs for specified ϵ and δ would be the locally similar time series within grey ribbons. Since two time series might be locally similar in more than one intervals, their matching subsequences are considered as two different pairs, one for each interval. For instance, in Figure 4 the green and red time series yield two matching pairs in different time intervals.

The above problem can be extended to the detection of groups, called *bundles*, of co-evolving time series. Each such bundle of time series contains at least a pre-defined number $\mu > 2$ of members, which are pairwise locally similar to each other over a time interval of sufficient duration. This problem can be formulated as follows:

PROBLEM 2 (LOCAL BUNDLE DISCOVERY OVER TIME SERIES). *Given a set of co-evolving time series $X = \{X_1, X_2, ..., X_n\}$ of equal length k,*
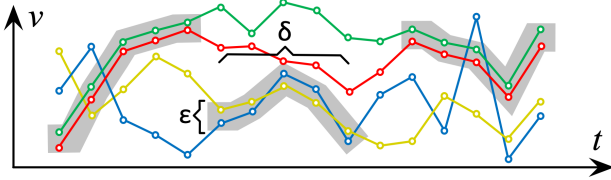
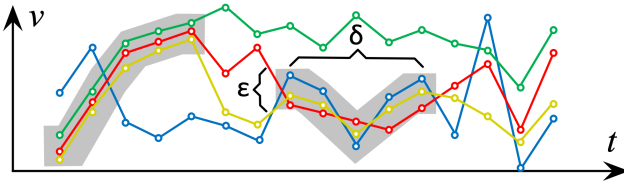Figure 4: Pair discovery over a set of time series.

Figure 5: Bundle discovery over a set of time series.

a minimum bundle size $\mu > 2$, $(\mu \in \mathbb{N})$, a maximum value difference $\epsilon > 0$, and a minimum time duration $\delta > 1$ timestamps, $(\delta \in \mathbb{N})$, retrieve all groups $\mathcal{G}$ of time series such that:

- *Each group $G \in \mathcal{G}$ contains at least μ time series.*
- *Within each group $G \in \mathcal{G}$, all pairs of time series are locally similar with respect to ϵ and δ.*
- *Each group $G \in \mathcal{G}$ is maximal, i.e., there is no other group $G' \supseteq G$ that also forms a bundle for the same time interval.*

An illustration of the above problem is shown in Figure 5. Each grey band covers the subsequences of at least $\mu = 3$ time series that constitute a bundle. These subsequences are pairwise locally similar for a specified distance ϵ and duration δ.

4 PAIR DISCOVERY

In this Section, we propose two solutions for the pair discovery problem. The first (Section 4.2) is a baseline algorithm that uses a sweep line to scan the co-evolving time series throughout their duration, while validating and keeping all the pairs that satisfy the given constraints employing a value discretization scheme per timestamp (Section 4.1). The second method (Section 4.4) employs an optimization that reduces the number of pairs to consider by judiciously probing candidates at selected timestamp values (referred to as *checkpoints*, Section 4.3). This significantly prunes the search space without missing any qualifying results.

4.1 Value Discretization

To reduce the candidate pairs that need be checked at each timestamp t, we discretize the values of all time series at t in *bins*, i.e., several consecutive value ranges, each one of size ϵ. Time series with values within the same bin at timestamp t form candidate pairs, but we also need to check adjacent bins for additional candidate pairs whose values differ by at most ϵ. Time series having values at non-adjacent bins are certainly farther than ϵ at that specific timestamp t, so we can avoid these checks.

To detect all candidate pairs and avoid cross-checking in every two adjacent bins we consider a *value range* of size ϵ, whose upper endpoint coincides with each value under consideration at time t.
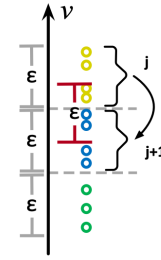
Figure 6: Discretization of time series values at timestamp t.

Then, all values of time series contained within this range, form candidate pairs (see Figure 6). Obviously, values contained in the same bin j can form candidate pairs. Then, we can cross-check each value in bin j with values in bin $j + 1$ for additional candidates with value difference at most ϵ, as indicated with the red (right) range.

At each timestamp t, the process of finding all the pairs consists of: (1) *Filtering* - Search among time series values in adjacent bins to detect candidate pairs using the aforementioned search method. (2) *Verification* - For each candidate pair, check similarity of their respective values at successive timestamps as long as this pair still qualifies to the matching conditions (or the end of time series data is reached). This step resembles to a "horizontal expansion" along the time axis in an attempt to eagerly verify and report pairs.

4.2 Pair Discovery Using Sweep Line

A baseline method for pair discovery over a set of time series is to check all the candidate pairs formed at each timestamp, and verify whether the minimum duration constraint δ is satisfied. Algorithm 1 describes this procedure. Pair discovery (Line 5) considers a time duration T (as a set of consecutive timestamps) to check for results. Initially $T = \{0, \ldots, k\}$, where k is the total duration of all time series data. For each timestamp t, we obtain only the subset of time series whose values are contained in two adjacent bins (Line 7). Based on these values at t, we obtain all candidate pairs with respect to the threshold ϵ (Line 9). For each such pair, if it is not already part of the resulting pairs at that specific timestamp t, we *verify* it by first expanding it horizontally (Lines 10-12) and checking whether this pair meets the duration constraint δ (Line 13) along subsequent timestamps. If so, we add it to the reported results (Line 14).

For the *horizontal expansion*, we iterate over the subsequent timestamps (after the current one – Line 17) and stop when ϵ is violated (Lines 18–19). Then, we mark the start of this pair with the current timestamp t, whereas its end is marked by the timestamp at which ϵ is crossed, and we return this pair (Lines 20-22).

However, searching over all timestamps in such an exhaustive manner can be expensive, particularly for long time series. Next, we present an optimization that identifies candidate pairs at selected timestamps only, so that only those pairs require verification.

4.3 Optimized Filtering at Checkpoints

To prune the search space, we consider *checkpoints* along the time axis, so that searching for candidate pairs will be performed at these specific timestamps only. If the temporal span between two successive checkpoints does not exceed the minimal duration threshold δ,

---

Algorithm 1: Sweep line scan pair discovery

Input: Set $\mathcal{X}$ of co-evolving time series of length k
Parameters: Threshold ϵ, min duration δ
**Output:** List P with all locally similar pairs of time series

1   $B \leftarrow CreateBins(\mathcal{X})$
2   $P \leftarrow DiscoverPairs(\emptyset, B, \{0, \ldots, k\}, \epsilon, \delta)$
3   **return** P

4   **Procedure** $DiscoverPairs(P, B, T, \epsilon, \delta)$
5   **foreach** $t \in T$ **do**
6   **foreach** $z = 0 \rightarrow B.size$ **do**
7         $\mathcal{X}' \leftarrow B_z^t \cup B_{z+1}^t$
8   **foreach** $X \in \mathcal{X}'$ **do**
9           $\mathcal{P}^t \leftarrow getAdjacentPairs(X^t, \epsilon)$
10   **foreach** $p \in \mathcal{P}^t$ **do**
11   **if** $p \notin P$ **then**
12               $p \leftarrow VerifyPair(p, t, k, \epsilon)$
13   **if** $p.end - p.start \geq \delta$ **then**
14                 $P \leftarrow P \cup p$

15   **return** P

16   **Procedure** $VerifyPair(p, t, k, \epsilon)$
17   **foreach** $t' = t + 1 \rightarrow k$ **do**
18   **if** $|p.X_i^{t'} - p.X_j^{t'}| > \epsilon$ **then**
19         $break$
20     $p.start \leftarrow t$
21     $p.end \leftarrow t'$
22   **return** p
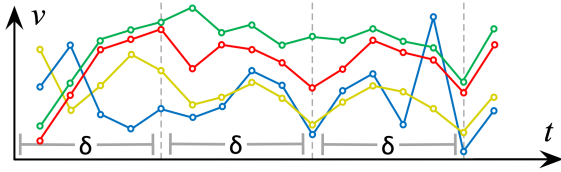
Figure 7: Checkpoints placed every δ timestamps.

we can ensure no false negatives, since any qualifying pair starting at an intermediate timestamp between two checkpoints will surely be detected at least on the second one. Figure 7 shows an example for a set of time series (checkpoints placed every $\delta = 5$ timestamps).

Assume a set of checkpoints placed at time interval δ from each other, as depicted in Figure 8(a). Let a checkpoint at timestamp t' and a qualifying pair of duration δ starting at timestamp $t' - \delta + 1$. This pair cannot have smaller duration, otherwise it would not meet constraint δ. Consequently, the pair will be detectable on the checkpoint at t', as shown in the figure. Similarly, if a qualifying pair ends at timestamp $t' + \delta - 1$ (Figure 8(b)), it will be detected at the checkpoint at t'. Hence, all pairs around a checkpoint at t' can be detected as candidates when we check their values at t'. Thus, we can easily conclude to the following observation.

LEMMA 1 (CHECKPOINT COVERING INTERVAL). *Let the interval between successive checkpoints not exceed δ. Considering a checkpoint placed at timestamp t', all qualifying pairs starting at $s, t' - \delta + 1 \leq s < t'$ and ending at $f, t' < f \leq t' + \delta - 1$ will satisfy all matching constraints at timestamp t'.*

This lemma entails that it suffices to check for candidate pairs only at checkpoints, i.e., every δ timestamps. We denote the set
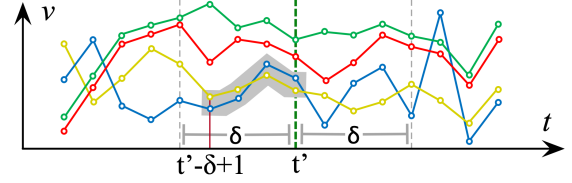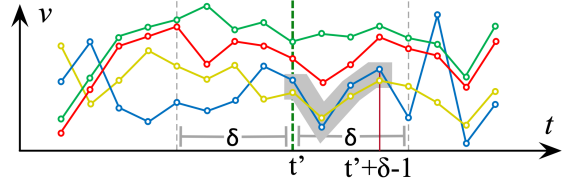

(a) Pair with a starting point before t'.

(b) Pair with an ending point after t'.

Figure 8: A qualifying pair will be detected on a checkpoint.
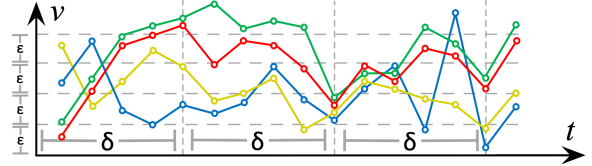

Figure 9: Sub-optimal checkpoint placement.

of checkpoints as C. Since we skip timestamps and in order to avoid false misses, we now have to verify pairs with a horizontal expansion (as in 4.2), but towards both directions, i.e., before and after a given checkpoint. Overall, at each checkpoint the optimized process performs: (1) *Filtering* - Search for candidate pairs among the values of time series in adjacent bins. (2) *Verification* - For each pair, perform a *two-way* horizontal expansion across the time axis.

Improving placement of checkpoints. Depending on the dataset, the default checkpoint placement might yield an *increased* number of candidate pairs, resulting in too many verifications. Intuitively, if the time series values at a specific timestamp t' are placed in a more *"scattered"* manner over the bins, less candidates would be generated. This is because the values of time series at t' would differ from each other by more than ϵ and thus can be pruned as described in Section 4.1. Figure 9 depicts such a case of sub-optimal placement of checkpoints, where the second checkpoint is placed at a rather *dense* area and as a result, six candidate pairs are considered. We can avoid this by shifting all checkpoints together either to the left or to the right, yet maintaining their temporal span every δ. As shown in Figure 10, all three checkpoints are collectively shifted to the left, avoiding the dense area and reducing the total number of candidate pairs. An extra checkpoint can be inserted before the first or after the last one, guaranteeing that there is no interval longer than δ without checkpoints.

Clearly, the placement of the set C of checkpoints influences the amount of candidate pairs. We wish to find the best such placement, which provides the least number of candidate pairs. The amount of candidates depends on the cardinality of the bins (i.e.,
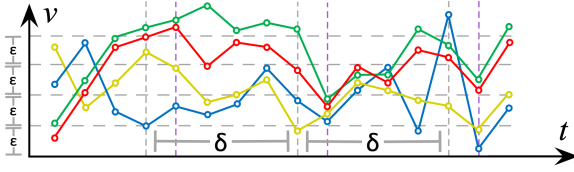
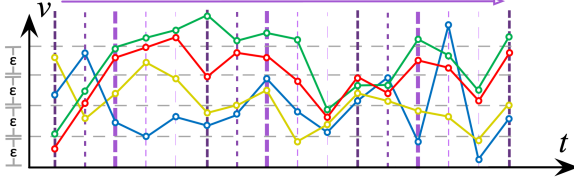**Figure 10: Improved checkpoint placement.**

Figure 11: Best checkpoint placement (thick vertical lines).

the number of values in each one, as shown in Fig. 6) at any particular checkpoint $c \in C$. Given that n is the total number of time series in the dataset (and hence the number of values at each checkpoint), we can identify the most populated bin at checkpoint c by calculating the maximal density $\frac{\max\{B_c\}}{n}$, where B_c represents the set of bin cardinalities at checkpoint c. Therefore, for a given configuration C of checkpoints, we can estimate an overall cost r by taking the sum of such maximal densities over all checkpoints, i.e., $r = \sum_{c \in C} \frac{\max\{B_c\}}{n}$. The less this total cost, the smaller the cardinality per bin at each checkpoint and, thus, the less the candidates that will be generated. Consequently, we seek to find the minimum r. To do so, we shift all checkpoints together to the right, one timestamp at a time, we estimate ratio r again and repeat δ times. This procedure is illustrated in Figure 11, where the checkpoints symbolized with similar lines belong to the same set as we move them to the right in order to identify the best placement (indicated with the thickest vertical dashed lines).

4.4 Pair Discovery Using Checkpoints

After identifying the best checkpoint placement, we can discover pairs of locally similar time series by applying the exhaustive algorithm presented in Section 4.2, but iterating over the defined checkpoints instead of all timestamps. To speed up the verification step, we introduce an optimization that reduces the number of checks. For each candidate pair p that started at (a possibly previous) timestamp $p.start$, we first expand its verification to the left. In case the current duration of pair p is still less than δ, we jump at timestamp $p.start + \delta$ in order to eagerly prune candidates that will not last at least δ. There, we check directly whether the values of these two time series qualify, and we continue this check backwards in time. If at an intermediate timestamp the ϵ constraint is not met, we can stop the verification process and discard the candidate pair.

The procedure for pair discovery is listed in Algorithm 2. Initially, we calculate the best possible checkpoint set (Line 2). Then, we run the procedure described in Section 4.2 but instead of probing over all timestamps we iterate over the resulting checkpoint set (Line 3).

Regarding verification, we first move towards the leftmost endpoint (Line 6) and detect the starting timestamp of the pair (Line 9). Then, we iterate from the timestamp $t_c = p.start + \delta$ towards

---

Algorithm 2: Checkpoint scan pair discovery

Input: Set X of co-evolving time series of length k
Parameters: Threshold ϵ, min duration δ
Output: A list P containing all the locally similar time series

1 $B \leftarrow CreateBins(X)$
2 $C \leftarrow GetCheckpoints(k, \delta)$
3 $P \leftarrow DiscoverPairs(\emptyset, B, C, \epsilon, \delta)$
4 **return** P

5 **Procedure** $VerifyPair2Way(p, t, k, \epsilon)$
6   **foreach** $t' = t - 1 \rightarrow 0$ **do**
7   **if** $|p.X_i^{t'} - p.X_j^{t'}| > \epsilon$ **then**
8     $break$
9   $p.start \leftarrow t'$
10 $t_c \leftarrow p.start + \delta - 1$
11 **foreach** $t' = t_c \rightarrow t$ **do**
12   **if** $|p.X_i^{t'} - p.X_j^{t'}| > \epsilon$ **then**
13    $p.f \leftarrow t$
14   **return** p
15 **foreach** $t' = t_c + 1 \rightarrow k$ **do**
16   **if** $|p.X_i^{t'} - p.X_j^{t'}| > \epsilon$ **then**
17    $break$
18 $p.end \leftarrow t'$
19 **return** p

---

the initial timestamp t (Line 11). If the pair does not qualify at a timestamp during this interval (Line 12), we set t as its final timestamp and return the pair (Line 14). Otherwise, we continue from timestamp t_c and towards the rightmost timestamp until the pair ceases to qualify (Lines 15-19).

Cost analysis. Let N_{tc} be the number of examined timestamps (i.e., checkpoints) considered by the algorithm. For each such timestamp, the algorithm needs to perform two operations, namely first to generate the set of candidate pairs and second to verify each pair. Let C_{cg}, C_{cv} and N_c denote, respectively, the candidate generation cost, the verification cost per candidate and the number of generated candidates. Then, the total cost is $C = N_{tc} \times (C_{cg} + N_c \times C_{cv})$. The candidate generation cost C_{cg} is proportional to the number of candidates N_c, which in turn is $O(|X|^2)$. However, in practice, the algorithm only needs to generate candidate pairs from the time series corresponding to the same bin. Let β denote the number of bins ($\beta \leq (y_{max} - y_{min})/\epsilon$) and N_β the maximum number of time series associated with any bin. Then, the expected cost in practice would be $\beta \times N_\beta^2$. As ϵ increases, β decreases but N_β increases (in the worst case, $N_\beta = |X|$, i.e., there is a single bin that contains all time series). Moreover, any candidate that has been already generated and verified in a previous timestamp need not be verified again, so the number of candidate pairs to be verified in the worst case is $O(|X|^2)$. Regarding the verification cost C_{cv}, consider a pair of time series (X_i, X_j) that is generated at timestamp τ. The algorithm needs to check for each subsequent timestamp $t \in (\tau, k)$ whether $|X_i^t - X_j^t| \leq \epsilon$, as long as this condition holds; hence, the cost is $O(k)$. Of course, in practice, this will also require much fewer comparisons in most cases. Notice that the difference between Algorithm 2 and Algorithm 1 is that the former generates and verifies candidates only at checkpoints, i.e., $N_{tc} = k/\delta$, while the latter at every timestamp, i.e., $N_{tc} = k$.

5 BUNDLE DISCOVERY

We now consider the bundle discovery problem. We propose two algorithms: an exhaustive one using a sweep line (Section 5.1), and one using checkpoints (Section 5.2), following the same principles as in pair discovery.

5.1 Bundle Discovery Using Sweep Line

To exhaustively detect bundles of time series, we can follow a similar procedure employing a sweep line as in Section 4.2. However, this time we detect candidate bundles at each timestamp before verifying whether constraints concerning minimal duration δ and minimum membership μ are satisfied. Essentially, this can be thought of as an adaptation of the flock discovery approach in [25] to the 1-dimensional setting in the case of time series.

Algorithm 3 describes this exhaustive process. Similarly to pair discovery, at each timestamp t (Line 5) we obtain only the values of time series contained in adjacent bins (Lines 6–7). Then, each such value is considered the origin of a search range ϵ, which returns a candidate group at time t (Line 9). Of course, such a candidate group may have been already included in the result bundles previously, during a horizontal expansion at a previous timestamp. In this case, its examination is skipped. Otherwise, if this group contains more than μ members, we proceed to verify it as a candidate bundle over subsequent timestamps via horizontal expansion (Lines 10-12). As will see next, this expansion may return one or more candidate bundles; each one is checked against the duration constraint δ before adding it to the result bundles (Lines 13-16).

Regarding the verification step of a candidate bundle $\mathcal{G}^t$, we apply its horizontal expansion over all subsequent timestamps (Line 19). For each member of such candidate bundle (Lines 21-28), we find the group $\mathcal{G}^{t'}$ of time series having values within range ϵ at time t'. Many such new groups may be created, as each member may yield one group. Such a group $\mathcal{G}^{t'}$ may become a new candidate bundle if it satisfies the μ constraint; if so, it is added to the resulting bundles with an updated duration (Lines 24–27). As we look at subsequent timestamps, it may happen that the same bundle may be added to the results multiple times, but with increasing duration. In the end, we eliminate duplicates and only keep the one with the longest duration (Line 28). Expansion stops once no new candidate bundles can be found in the next timestamp (Lines 29–30).

5.2 Bundle Discovery Using Checkpoints

For bundle discovery using checkpoints, we apply a similar sweep line approach, but this time we only filter at checkpoints and then verify towards both directions in the time axis. Algorithm 4 describes this procedure. After initializing the checkpoints (Line 2), we run the bundle discovery process as in Algorithm 3, but this time looking at checkpoints (Line 3) instead of all timestamps. For the two-way horizontal expansion, we first examine all candidate bundles in set Q_1 detected from the current checkpoint towards the origin of time axis (Line 7). This is done because a qualifying bundle could have started earlier, before the current checkpoint. Afterwards, we apply the same eager pruning strategy as in Section 4.4. So, we verify each such candidate bundle jumping forward at timestamp $t_c = q.start + \delta$ and continue backwards in time to its currently known start (Lines 8–10). Among the candidate bundles

---

Algorithm 3: Sweep line scan bundle discovery

Input: Set $\mathcal{X}$ of co-evolving time series of length k
Parameters: Threshold ϵ, min duration δ, min members μ
Output: A list P containing all the discovered bundles

1   $B \leftarrow CreateBins(\mathcal{X})$
2   $P \leftarrow DiscoverBundles(\emptyset, B, \{0, ..., k\}, \epsilon, \delta, \mu)$
3   **return** P

4   **Procedure** $DiscoverBundles(P, B, T, \epsilon, \delta, \mu)$
5   **foreach** $t \in T$ **do**
6   **foreach** $z \rightarrow B.size$ **do**
7         $\mathcal{X}' \leftarrow B_z^t \cup B_{z+1}^t$
8   **foreach** $X \in \mathcal{X}'$ **do**
9           $\mathcal{G}^t \leftarrow getAdjacent(X^t, \epsilon)$
10   **if** $\mathcal{G}^t \notin P$ **then**
11   **if** $\mathcal{G}^t.size \geq \mu$ **then**
12               $\mathcal{B} \leftarrow VerifyBundle(\mathcal{G}^t, \{t, .., k\}, \epsilon, \mu)$
13   **foreach** $\mathcal{G} \in \mathcal{B}$ **do**
14   **if** $\mathcal{G}.end - \mathcal{G}.start \geq \delta$ **then**
15                   $P \leftarrow P \cup \mathcal{G}$

16   **return** P

17   **Procedure** $VerifyBundle(\mathcal{G}^t, \{t_1, ..., t_\phi\}, \epsilon, \mu, f)$
18     $P \leftarrow \mathcal{G}^t$
19   **foreach** $t' = t_2 \rightarrow t_\phi$ **do**
20       $expanded \leftarrow False$
21   **foreach** $\mathcal{G} \in P$ **do**
22   **foreach** $X \in \mathcal{G}$ **do**
23           $\mathcal{G}^{t'} \leftarrow getAdjacent(X^{t'}, \epsilon)$
24   **if** $(\mathcal{G}^{t'}.size \geq \mu$ **then**
25   Rearrange duration of $\mathcal{G}^{t'}$ accordingly
26             $P \leftarrow P \cup \mathcal{G}^{t'}$
27             $expanded \leftarrow True$
28       $P \leftarrow keepLongestBundles(P)$
29   **if** $expanded = False$ **then**
30         $break$

31   **return** P

Algorithm 4: Checkpoint scan bundle discovery

Input: Set $\mathcal{X}$ of co-evolving time series of length k
Parameters: Threshold ϵ, min duration δ, min members μ
Output: A list P containing all the discovered bundles

1   $B \leftarrow CreateBins(\mathcal{X})$
2   $C \leftarrow GetCheckpoints(k, \delta)$
3   $P \leftarrow DiscoverBundles(\emptyset, B, C, \epsilon, \delta, \mu)$
4   **return** P

5   **Procedure** $VerifyBundle2Way(\mathcal{G}^t, t, k, \epsilon, \mu)$
6     $P, Q_1, Q_2 \leftarrow \emptyset$
7     $Q_1 \leftarrow VerifyBundle(\mathcal{G}^t, \{t-1, ..., 0\}, \epsilon, \mu)$
8   **foreach** $q \in Q_1$ **do**
9       $t_c \leftarrow q.start + \delta - 1$
10       $Q_2 \leftarrow Q_2 \cup VerifyBundle(q, \{t_c, ..., q.start\}, \epsilon, \mu)$
11   **foreach** $q \in Q_2$ **do**
12   **if** $q.end - q.start \geq \delta$ **then**
13         $P \leftarrow P \cup VerifyBundle(q, \{t_c, ..., k\}, \epsilon, \mu)$
14   **return** P

(in set Q_2) returned from the forward verification (Line 11), we care only for those that satisfy the minimal duration constraint δ (Line 12). These are further verified from timestamp t_c and forward in time, obtaining all subsequent qualifying bundles (Line 13).

Table 1: Datasets used in the experiments.

Dataset	Size	Time series length
Water	822	168
Synthetic	50,000	1,000

Cost analysis. The algorithm follows a similar approach as for the case of pairs. At selected timestamps, first the candidate bundles are generated, and then each bundle is verified. In this case, since we are seeking groups of at least m time series, the number of candidates N_c is $O(|X|^\mu)$, although in practice it is again expected to be much lower since only those time series corresponding to the same bin need to be considered. Finally, verification is similar with only slightly higher cost. Indeed, when checking the condition $|X_i^t - X_j^t| \leq \epsilon$, we first need to determine the time series X_i and X_j inside the bundle that have the highest and lowest values, respectively.

6 EXPERIMENTAL EVALUATION

6.1 Experimental Setup

We evaluate the performance of our methods on pair and bundle discovery both qualitatively and quantitatively. We compare our checkpoint (CP) scan approaches for each problem with the respective sweep line (SL) methods. We use a real-world and a synthetic dataset as listed in Table 1. Next, we describe their characteristics.

DAIAD Water Consumption (Water). Courtesy of the DAIAD project[1], we acquired a time series dataset of hourly water consumption for 822 households in Alicante, Spain from 1/1/2015 to 20/1/2017. In order to get a more representative dataset for our tests, we first calculated the weekly time series (24×7 timestamps, one value per hour from Monday to Sunday) per household by averaging corresponding hourly values over the entire period.

Synthetic Dataset. We generated a synthetic dataset of 50,000 time series, each with a length of 1,000 timestamps. So, this dataset contains 50 million data points in total. The dataset was generated in a similar manner to the synthetic dataset used in [11].

All experiments were conducted on a Dell PowerEdge M910 with 4 Intel Xeon E7-4830 CPUs, each containing 8 cores clocked at 2.13GHz, 256 GB RAM and a total storage space of 900 GB.

6.2 Evaluation Results

We conducted two sets of experiments, using the water and synthetic datasets. The water dataset was used for qualitative and quantitative assessment of our methods on pair and bundle discovery, while the synthetic dataset was used for efficiency evaluation.

6.2.1 Pair and Bundle Discovery over Real Data. We performed several experiments using the water dataset for various parameter values to detect pairs and bundles using both the SL and CP approaches. The dataset was *z-normalized* to eliminate amplitude discrepancies among time series and focus on structural similarity.

To evaluate our methods against different parameters, we performed preliminary tests to extract ranges of values where the algorithms would return a reasonable number of results. Table 2 lists the range of values for the parameters used for bundle and pair discovery tests (recall that parameter μ is not applicable in pair discovery); default values are in bold. Parameter δ is expressed as

Table 2: Parameters for tests over the water dataset

Parameter	Values
δ (% of time series length, i.e., 168)	6%, 5%, **6%**, 7%, 8%
ϵ (% of value range, i.e., approx 11.4)	4%, 5%, **6%**, 7%, 8%
μ (% of dataset size, i.e., 822)	0.5%, 0.75%, **1%**, 1.25%, 1.5%

a percentage of the duration of the time series, ϵ is expressed as a percentage of the value range (i.e., difference $max - min$ in values encountered across the dataset) and μ is expressed as a percentage of the number of time series in the dataset.
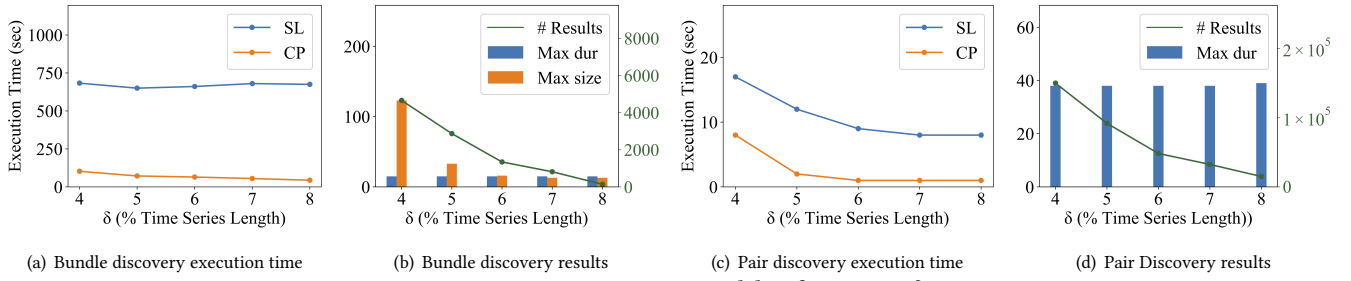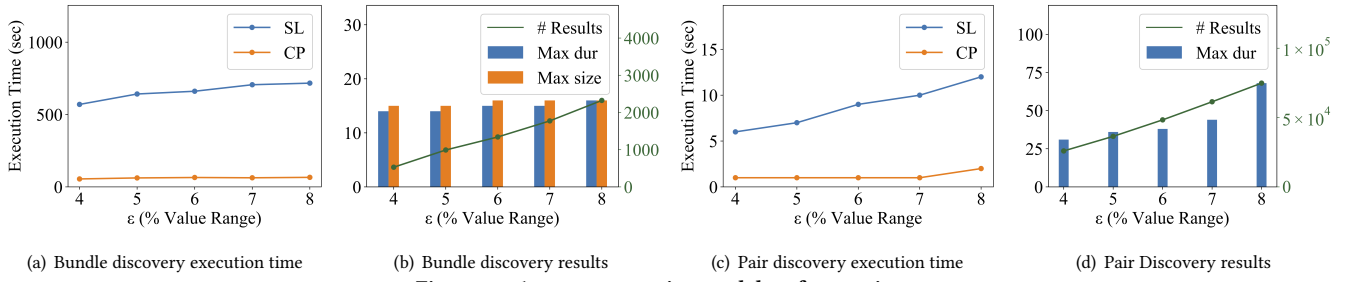
Varying δ. Figure 12 depicts the results for varying minimal duration δ. In bundle discovery, the CP algorithm outperforms SL up to an order of magnitude in terms of execution time (Figure 12(a)). As the threshold δ gets larger, performance is improved due to less checkpoints being specified and less candidates needing verification. On the contrary, the SL approach performs similarly irrespective of δ, as time series must be checked at all timestamps. From Figure 12(b), it turns out that the number of detected bundles is reduced as the δ value increases, which is expected as fewer bundles can last longer. In this plot, the blue bars indicate the maximum bundle duration among the ones that were detected, while the orange bars indicate the larger detected bundle in terms of membership. It is clear that the maximum bundle size is drastically reduced as the number of results diminish, while the maximum duration among bundles remains the same with the increase of δ, as the longest bundle is the same in these results.

Regarding pair discovery, since it is an overall faster process, the differences in terms of efficiency are smaller, but still apparent. In this case, the execution time (Figure 12(c)) is more abruptly reduced in both SL and CP methods, since less subsequences qualify as pairs. The number of results (Figure 12(d)) is now naturally much larger, as far more pairs are expected to be verified if bundles exist. The same stands for the maximum duration among pairs, which tend to last longer compared to bundles. Since a pair is actually a bundle with μ=2 members, it is easier to find local similarity over longer intervals between two subsequences rather than an increased number of them. The maximum duration, as in bundle discovery, remains the same as δ increases, since this corresponds to the same pair in the results.

Varying ϵ. Varying threshold ϵ for bundle discovery slightly incurs more execution cost for both SL and CP approaches. This is due to the increased number of bundles that need to be verified. Nonetheless, the difference in cost remains at levels of at most an order of magnitude, as shown in Figure 13(a). As expected, the number of results is also increased (Figure 13(b)). So does the maximum duration bundle, which is also expected due to more qualifying bundles, hence a higher probability to find longer ones. The maximum bundle size (i.e., membership) is also increased, as more time series can form a bundle when allowing a wider threshold ϵ in deviation of their respective values.

Regarding pair discovery, the results are again similar to bundle discovery for varying ϵ. For very small ϵ values of up to 7% of the value range, the CP algorithm returns results almost instantly. The SL approach is at least five times slower, with its performance deteriorating more rapidly with increasing ϵ values. Again, as in bundle discovery, the results are growing with greater ϵ, as does

(a) Bundle discovery execution time (b) Bundle discovery results (c) Pair discovery execution time (d) Pair Discovery results

Figure 12: Assessment against real data for varying δ.

(a) Bundle discovery execution time (b) Bundle discovery results (c) Pair discovery execution time (d) Pair Discovery results

Figure 13: Assessment against real data for varying ϵ.

the maximum duration among pairs, especially for ϵ equal to 8% of the value range.

Varying μ. When varying the minimum membership parameter μ in bundle discovery (Figure 14), the results regarding execution time are again very similar to the rest of the tests as indicated in Figure 14(a). Again, the CP algorithm outperforms SL up to one order of magnitude. The execution time is very slightly decreased for larger μ values in both algorithms, as more candidate bundles are pruned. Results from CP are reported almost instantly, since the number of time series is rather small and scanning through the limited number of checkpoints is very fast. This explains why performance of SL does not get drastically improved as μ gets larger, since filtering and verification has to be repeated at every timestamp. The number of results (Figure 14(b)) is reduced as μ increases, which is expected, as less bundles get detected with a larger membership. The maximum duration among bundles also decreases; interestingly, the maximum size detected among bundles increases as the number of results diminishes, due to the growing number μ of required number of members per bundle.

6.2.2 Efficiency against Synthetic Data. To evaluate the efficiency of our methods, we used the synthetic dataset. Regarding parameter
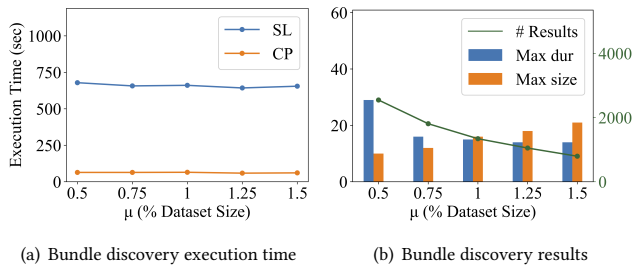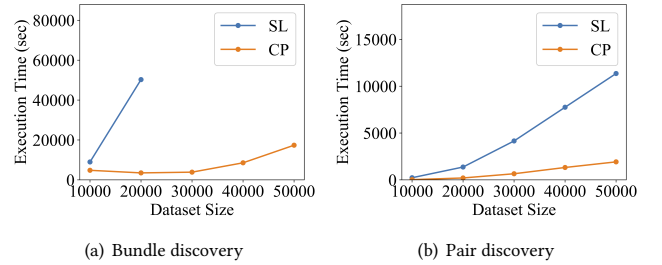

(a) Bundle discovery execution time (b) Bundle discovery results

Figure 14: Assessment against real data for varying μ.

(a) Bundle discovery (b) Pair discovery

Figure 15: Efficiency with varying numbers of time series.

Table 3: Parameters for tests against synthetic data

Parameter	Values
Dataset Size	**10000**, 20000, 30000, 40000, 50000
Time Series Length	600, 700, **800**, 900, 1000
δ (% of time series length)	**2.5%**
ϵ (% of value range)	**0.2%**
μ (% of dataset size)	**1.25%**

values, as in the previous experiments, we performed preliminary tests to extract ranges of values where the algorithms return a reasonable number of results. Table 3 lists the range of values for all parameters used in these efficiency tests for bundle and pair discovery, with the default values emphasized in bold (again, μ is not applicable in pair discovery).

Varying Dataset Size. Figure 15 depicts the performance comparison between CP and SL algorithms for bundle and pair discovery. We omit cases where execution of an algorithm was taking more than 15 hours (cutoff). As illustrated in Figure 15(a), an increase in the dataset size leads to a very abrupt deterioration of performance for the SL algorithm of up to several hours of execution for 20,000 time series. For larger dataset sizes, the execution time was significantly longer than the cutoff time. On the other hand, CP reports
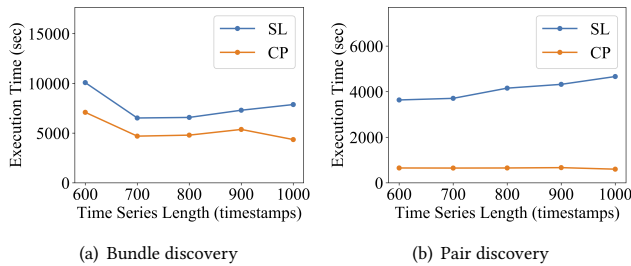
(a) Bundle discovery   (b) Pair discovery

Figure 16: Efficiency with varying length of time series.

results in all cases. Its execution time increases for larger dataset sizes, but manages to finish in a few hours in the worst case (for 50,000 time series). It is worth noting that membership parameter μ is more relaxed for larger dataset sizes, as it is expressed in terms of percentage of the total number of time series in the dataset. This explains the slight improvement in performance for dataset sizes of 20,000 and 30,000 for the CP method. In general, CP is more than an order of magnitude faster than SL, which also stands for the case of pair discovery, as illustrated in Figure 15(b). As the number of time series in the dataset grows, it is natural that more pairs will be detected, hence the linear increase in the execution cost for the CP algorithm. Of course, baseline SL requires more time, as it must check many more combinations of time series.

Varying Time Series Length. For time series with increasing length (Figure 16(a)), the CP algorithm for bundle discovery again constantly outperforms SL. Similarly to previous experiments, δ is expressed as a percentage of the time series length. We observe that the execution time initially decreases for both algorithms, as more bundles are pruned. However, as the time series length (and δ) gets larger, the performance of both algorithms slightly worsens. Only in the case of 1,000 timestamps the execution time starts to drop for the CP algorithm due to the even larger δ. This is not the case with the SL method, which has to evaluate more timestamps. Notice that the difference between the SL and CP algorithms is smaller, compared to the real dataset. This is due to the larger number of existing bundles in the synthetic dataset, which were detected on the checkpoints and had to be verified. Similar observations stand for pair discovery, with the CP algorithm significantly outperforming SL in all cases (Figure 16(b)).

7 CONCLUSIONS

In this paper, we addressed the problems of pair and bundle discovery over co-evolving time series, according to their local similarity. We introduced two efficient algorithms for pair and bundle discovery that utilize checkpoints appropriately placed across the time axis in order to aggressively prune the search space and avoid expensive exhaustive search at each timestamp. Our methods successfully detect locally similar subsequences of co-evolving time series, as demonstrated in our experimental evaluation over real-world and synthetic data. Also, they were orders of magnitude faster compared to baseline methods that apply a sweep line approach, confirming their effectiveness in pair and bundle discovery. In the future, we plan to further improve the scalability of our algorithms to extend their applicability over very large time series datasets, both in terms of cardinality, as well as in terms of length.

REFERENCES

[1] Marc Benkert, Joachim Gudmundsson, Florian Hübner, and Thomas Wolle. 2008. Reporting flock patterns. *Computational Geometry* 41, 3 (2008), 111–125.
[2] Yongjie Cai, Hanghang Tong, Wei Fan, and Ping Ji. 2015. Fast mining of a network of coevolving time series. In *SDM*. 298–306.
[3] Alessandro Camerra, Themis Palpanas, Jin Shieh, and Eamonn J. Keogh. 2010. iSAX 2.0: Indexing and Mining One Billion Time Series. In *ICDM*.
[4] Alessandro Camerra, Jin Shieh, Themis Palpanas, Thanawin Rakthanmanon, and Eamonn J. Keogh. 2014. Beyond one billion time series: indexing and mining very large time series collections with iSAX2+. *KAIS* 39, 1 (2014), 123–151.
[5] Kin-pong Chan and Ada Wai-Chee Fu. 1999. Efficient Time Series Matching by Wavelets. In *ICDE*. 126–133.
[6] Richard Cole, Dennis Shasha, and Xiaojian Zhao. 2005. Fast window correlations over uncooperative time series. In *SIGKDD*. 743–749.
[7] Rui Ding, Qiang Wang, Yingnong Dang, Qiang Fu, Haidong Zhang, and Dongmei Zhang. 2015. YADING: Fast Clustering of Large-scale Time Series Data. *Proc. VLDB Endow.* 8, 5 (Jan. 2015), 473–484.
[8] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2018. The Lernaean Hydra of Data Series Similarity Search: An Experimental Evaluation of the State of the Art. *PVLDB* 12, 2 (2018), 112–127.
[9] Amara Graps. 1995. An Introduction to Wavelets. *IEEE Comput. Sci. Eng.* 2, 2 (1995), 50–61.
[10] Joachim Gudmundsson and Marc van Kreveld. 2006. Computing longest duration flocks in trajectory data. In *ACM GIS*. 35–42.
[11] Eamonn J Keogh and Michael J Pazzani. 1999. An indexing scheme for fast similarity search in large time series databases. In *SSDBM*. 56–67.
[12] Haridimos Kondylakis, Niv Dayan, Kostas Zoumpatianos, and Themis Palpanas. 2018. Coconut: A Scalable Bottom-Up Approach for Building Data Series Indexes. *PVLDB* 11, 6 (2018), 677–690.
[13] Jessica Lin, Eamonn J. Keogh, Li Wei, and Stefano Lonardi. 2007. Experiencing SAX: a novel symbolic representation of time series. *DAMI* 15, 2 (2007), 107–144.
[14] Michele Linardi and Themis Palpanas. 2018. Scalable, variable-length similarity search in data series: The ULISSE approach. *PVLDB* 11, 13 (2018), 2236–2248.
[15] Yasuko Matsubara, Yasushi Sakurai, and Christos Faloutsos. 2014. Autoplait: Automatic mining of co-evolving time sequences. In *SIGMOD*. ACM, 193–204.
[16] Themis Palpanas. 2015. Data Series Management: The Road to Big Sequence Analytics. *SIGMOD Record* 44, 2 (2015), 47–52.
[17] Themis Palpanas. 2016. Big Sequence Management: A glimpse of the Past, the Present, and the Future. In *SOFSEM*. 63–80.
[18] Spiros Papadimitriou, Jimeng Sun, and S Yu Philip. 2006. Local correlation tracking in time series. In *ICDM*. 456–465.
[19] John Paparrizos and Luis Gravano. 2015. k-Shape: Efficient and Accurate Clustering of Time Series. In *SIGMOD*. 1855–1870.
[20] John Paparrizos and Luis Gravano. 2017. Fast and Accurate Time-Series Clustering. *ACM Trans. Database Syst.* 42, 2 (2017), 8:1–8:49.
[21] Botao Peng, Panagiota Fatourou, and Themis Palpanas. 2018. ParIS: The Next Destination for Fast Data Series Indexing and Query Answering. In *IEEE BigData*.
[22] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. 2012. Searching and mining trillions of time series subsequences under dynamic time warping. In *SIGKDD*. 262–270.
[23] Jin Shieh and Eamonn J. Keogh. 2008. *i*SAX: indexing and mining terabyte sized time series. In *SIGKDD*. 623–631.
[24] Pedro Sena Tanaka, Marcos R Vieira, and Daniel S Kaster. 2015. Efficient Algorithms to Discover Flock Patterns in Trajectories.. In *GeoInfo*. 56–67.
[25] Marcos R Vieira, Petko Bakalov, and Vassilis J Tsotras. 2009. On-line discovery of flock patterns in spatio-temporal data. In *SIGSPATIAL*. 286–295.
[26] Djamel-Edine Yagoubi, Reza Akbarinia, Florent Masseglia, and Themis Palpanas. 2018. Massively Distributed Time Series Indexing and Querying. *TKDE (to appear)* (2018).
[27] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, and Eamonn Keogh. 2016. Matrix profile I: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In *ICDM*.
[28] Yunyue Zhu and Dennis Shasha. 2002. Statstream: Statistical monitoring of thousands of data streams in real time. In *VLDB*. 358–369.
[29] Kostas Zoumpatianos, Stratos Idreos, and Themis Palpanas. 2014. Indexing for interactive exploration of big data series. In *SIGMOD*. 1555–1566.