Mining Anomalies in Subspaces of High-dimensional Time Series for Financial Transactional Data

Jingzhu He ✉[1], Chin-Chia Michael Yeh[2], Yanhong Wu[2], Liang Wang[2], and Wei Zhang[2]

[1] North Carolina State University, jhe16@ncsu.edu
[2] Visa Research, {miyeh,yanwu,liawang,wzhan}@visa.com

Abstract. Anomaly detection for high-dimensional time series is always a difficult problem due to its vast search space. For general high-dimensional data, the anomalies often manifest in subspaces rather than the whole data space, and it requires an $O(2^N)$ combinatorial search for finding the exact solution (i.e., the anomalous subspaces) where N denotes the number of dimensions. In this paper, we present a novel and practical unsupervised anomaly retrieval system to retrieve anomalies from a large volume of high dimensional transactional time series. Our system consists of two integrated modules: subspace searching module and time series discord mining module. For the subspace searching module, we propose two approximate searching methods which are capable of finding quality anomalous subspaces orders of magnitudes faster than the brute-force solution. For the discord mining module, we adopt a simple, yet effective nearest neighbor method. The proposed system is implemented and evaluated on both synthetic and real-world transactional data. The results indicate that our anomaly retrieval system can localize high quality anomaly candidates in seconds, making it practical to use in a production environment.

Keywords: Unsupervised Anomaly Retrieval · High-dimensional Time Series · Subspace Searching · Data Mining.

1 Introduction

Time series anomaly detection is important for building automatic monitoring systems. Although anomaly detection in time series data has been extensively studied in literature for decades, the majority of prior work only detects anomalies on either one or all dimensions. While searching anomalies in subspaces is commonly studied in vector space-based methods [4, 11], nearly no work has been done in searching anomalies in subspaces of multidimensional time series. If the system does not retrieve anomalies from the correct subspace, it often results in producing undesirable results due to false dismissals similar to the case of multidimensional motif discovery [35].

Let us consider an example as shown in Fig. 1 where we have a three-dimensional time series and aim to identify the days containing anomalies. The state-of-the-art discord mining-based methods [4, 17] generally compare the distances between time series associated with each pair of days and generate anomaly alerts once the nearest neighbor distances exceed certain thresholds. If we apply these methods independently

on each dimension (dim. 0, 1, or 2), as every daily pattern occurs twice, all nearest neighbor distances between the days are low and no alert will be generated on any of these dimensions. These results also hold if we apply the same algorithm to the combined time series (all dim.). Only if we combine dimension 0 and dimension 1 (dim. 0 + 1), the anomaly, which occur on day one, can be detected by the discord mining-based method. In other words, the anomaly detection system will falsely dismiss the anomalies if it does not exhaustively search anomalies in *all* possible combinations of dimensions (i.e., subspaces).

Detecting anomalies in subspaces is crucial in many domains. In this paper, we focus on finding anomalies in financial transaction data, a particular area where a failed detection strategy may cause multi-million-dollar losses. For example, during the 2013 ATM cyber looting attack, US$2.4 million was looted from about 3,000 ATMs in New York City [27]. The attackers evenly distributed looting to the targeted ATMs so that only an extra US$800 was withdrawn from each ATM. Given that such a small amount of perturbation can be considered as a normal daily fluctuation, this attack is not discoverable by monitoring ATM's associated time series[3] individually. Meanwhile, this attack can neither be detected by monitoring the aggregated transaction volume of all the ATMs in the U.S. as the targeted ATMs represent only 5% of the total ATMs [32]. Although some pre-defined rules (e.g., detecting withdrawals with the same dollar amount that occurred at multiple ATMs within a short time) can capture the anomalies in this case, attackers can quickly learn the rules and adapt their behaviors accordingly. Therefore, we need to develop an algorithm to detect the "correct" combinations of time series associated with each ATM without relying on simple rules.

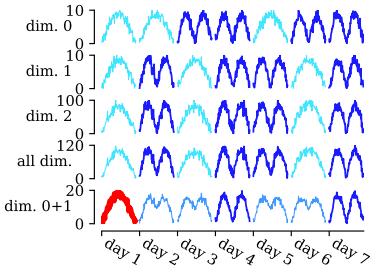

Fig. 1. The anomalous patterns (red/bold) are detectable only when dimension 0 and dimension 1 are combined. The blue/thin lines are recurrent (i.e., normal) patterns.

A simple solution is exhaustive search (i.e., brute-force), by examining anomalies in all possible subspaces (i.e., combination of dimensions), potential anomalies manifested in subspaces can be identified. However, such brute-force solution is not scalable: searching all possible dimension combinations requires an $O(2^N)$ time complexity where N denotes the number of dimensions. Such complexity is infeasible for most real-world data, especially for transactional data. Collectively, a typical global payment company generates hundreds of millions of transaction records per day and each transaction record is typically associated with hundreds of attributes, representing different characteristics, such as transaction type, zipcode, and transaction amount. By aggregating statistics for transaction records associated with different common categorical attributes (e.g., a particular zipcode) hourly, we can generate *multidimensional time series* from transactional data. As anomalies may manifest in subspaces of the multidimensional time series,

[3] The time series is generated by hourly withdrawn amount.

the anomaly retrieval system needs to quickly identify the potential most suspicious subspaces.

This paper presents a novel unsupervised anomaly retrieval system on multidimensional time series data. We design the anomaly retrieval system with two modules, i.e., *subspace searching* and *time series discord mining*. We present and evaluate two alternative approaches to perform approximate subspace search, i.e., *greedy search* and *evolutionary search*. The proposed approximate subspace searching methods are capable of finding quality anomalies with their runtime being an order of magnitude faster than the brute-force solution. For time series discord mining, we design a nearest neighbor-based method with Dynamic Time Warping (DTW) to locate and score the anomalies. We outline two different advanced implementations of the discord mining method with 86% runtime reduction over the naive implementation. Adopting improvements proposed for both modules, the proposed system is practical for deployment in the production environment for monitoring transactional data. Our paper makes the following contributions.

1. We investigate the unsupervised anomaly retrieval problem on multidimensional time series. We divide the problem into two sub-problems and design an anomaly retrieval system with two modules: subspace searching and discord mining.
2. We propose two different approaches to find the most anomalous subspace from an $O(2^N)$ search space for financial transactional data.
3. We design an efficient discord mining method based on DTW distances to identify the temporal location of the anomalies and evaluate the anomalous degree of the anomalies.
4. We implement our algorithm and conduct comprehensive experiments on both synthetic data and real-world transactional data. The experiment results show that our system outperforms all alternative approaches and it is practical for applications in real-world products.

2 Related Work

The anomaly detection problem has been extensively studied for over a decade, and there are many variants of the problem [2, 4, 11, 12]. In this section, we focus on two variants that are mostly relevant to this work: *high-dimensional data anomaly detection* and *time series anomaly detection*.

High-dimensional data anomaly detection: The proposed methods for high-dimensional data anomaly detection problem usually attempt to solve the curse of dimensionality associated with high-dimensional data and find anomalies that manifest in subspace span by a subset of dimensions [12]. These methods either use alternative anomaly scores to combat the curse of dimensionality or define/search for a subspace where the anomalies are most likely to manifest. For example, angle-based methods solve the curse of dimensionality by mining anomalies based on angles between the vector space representation of the data instead of Euclidean distance [12, 18, 38, 39]. The hyperplane-based method proposed in [19] defines subspaces with respect to each data point's neighbors. The

UBL [5] method performs anomaly detection on all dimensions of system metrics based on neural networks. To search the subspaces where anomalies may manifest, various approaches have been explored such as bottom-up search [23, 34], dependency among different dimensions [16], dimensionality unbiasedness [31], set enumeration tree [6], evolutionary algorithm [1] and domain knowledge [33]. Nevertheless, existing work typically identifies anomalous records in a database using the associated attributes represented as high-dimensional feature vectors. Even in the works that deal with streaming data [37–39], time series view of the data is not considered for the anomaly detection problem. To the best of our knowledge, our work is the first one to adopt time series representation of high dimensional data for anomaly detection.

Time series anomaly detection: Various techniques like Markov models, dynamic Bayesian networks, and neural networks are explored for time series anomaly detection [11], and different techniques are proposed to capture different types of anomalies. For example, Siffer el al. [29] designed an extreme value detection system based on extreme value theory for streaming time series data without requiring any manually-determined thresholds. Both DILOF [24] and MiLOF [26] detect time series anomaly based on the Local Outlier Factor (LOF) scores. Existing work such as [21] and [10] defines the anomaly based on data density. Another simple yet effective definition for time series anomaly is time series discord [4, 17]. It defines time series anomalies as the most unusual subsequences (subsets of consecutive data points) in time series. Besides these studies, many efforts have been made to apply deep learning-based anomaly detection on time series in various domains [3, 8, 15]. Malhotra et al. [20] and Su et al. [30] detect time series anomalies based on the reconstructed error computed from recurrent neural networks. The TScope method [13] adopts a unique feature extraction method and a customized Self-Organizing Map-based score to detect anomalies in system call traces. Most of the aforementioned work either considers each data point independently [10, 21, 24, 26, 29], or does not consider the fact that anomalies could manifest in subspace instead of full space [3, 8, 13, 15, 17, 20, 30]. As a result, to the best of our knowledge, our method is the only method that is capable of identifying anomalies based on time series discord definition in high-dimensional data.

3 Definitions and Notation

Definition 1. (Transaction record). Each record is formulated as $D = [d_1, d_2, d_3, ..., d_n, t, a]$, where each d_i represents a discrete attribute that has a finite set of possible values. n is the number of discrete attributes. Besides these discrete attributes, each transaction record has a timestamp t indicating the occurring time of the transaction and the transaction amount a, which is a numerical value.

Definition 2. (Transaction database). A transaction database **D** stores a collection of transaction records.

An example of a transaction database, consisting of four transaction records, is shown in Table 1. Three individuals (Alice, Bob, and Carlos) have transactions at three different merchants (eBay, Amazon, and Walmart) in two different states (California and Washington). To account for temporal variations of the transaction database, we generate a time series from the database using a *sliding aggregator*.

Table 1. Four records from an example transaction database.

Customer	Merchant	Location	Timestamp	Amount
Alice	eBay	CA	1559854527	35
Bob	eBay	CA	1559854635	35
Alice	Amazon	WA	1559854800	50
Carlos	Walmart	CA	1559859053	38

Definition 3. (Sliding aggregator). Given a transaction database $\mathbf{D}$, a sliding aggregator $A(\cdot)$ is an aggregating function that generates a time series by summarizing the statistics of transactions satisfied given conditions with a sliding window of window size w and hop size h.

For example, we want to look at the per hour transaction amounts in California for a database D. Then we apply the sliding aggregator $A(count(), w = 1\,\text{hour}, h = 1\,\text{hour}, Location = \texttt{CA})$ on D. Using the example database shown in Table 1, the first two transaction records, i.e., Alice and Bob, are fed into the $count()$ function as they satisfy the condition $Location = \texttt{CA}$ and occur within the same hour-long window. The function output is a time series that includes the aggregated transaction amounts of each sliding window.

Applying a single sliding aggregator only creates one view of the transaction database. To represent the database more holistically, we define the subspace set of a given transaction database as follows.

Definition 4. (Subspace set). Given a transaction database $\mathbf{D}$, the corresponding subspace set $\mathbf{S_D} = [T_1, T_2, T_3, ..., T_m]$ is a set of m univariate time series, where each $T_i \in \mathbf{S_D}$ is generated by applying different sliding aggregators and is considered as one of m subspaces.

As each $T_i \in \mathbf{S_D}$ stores one view of the database D, we regard each T_i as a subspace of $\mathbf{S_D}$. Continuing with the example shown in Table 1, by simply calculating the hourly counts for *all* combinations of locations, the total number of subspaces (i.e., number of $T_i \in \mathbf{S_D}$) will be $2^{50} - 1 \approx 1\,\text{quadrillion}$. To help us explain our subspace search methodology, we further define the concept of *unit subspace*.

Definition 5. (Unit subspace). Given a subspace set $\mathbf{S_D}$, a unit subspace is a subspace that cannot be obtained through a combination of other subspaces within $\mathbf{S_D}$.

Let us say that the $\mathbf{S_D}$ consists of the aforementioned one quadrillion subspaces generated by selecting all combinations of locations, a unit subspace is a subspace associated with a single location. For instance, the subspace $T_{CA} = A(Location = \texttt{CA})$[4] is a unit subspace while the subspace $T_{CA,WA} = A(Location = \texttt{CA} \vee Location = \texttt{WA})$ is *not* a unit subspace as $T_{CA,WA}$ can be generated by combining T_{CA} and T_{WA}. Throughout the paper, we use the term "dimension" and "unit subspace" interchangeably.

[4] Other inputs (i.e., $count()$, $w = 1\,\text{hour}$ and $h = 1\,\text{hour}$) of $A()$ are omitted for brevity.

Therefore, each dimension of the multivariate time series shown in Fig. 1 is a unit subspace, and the overall subspace set includes both the unit subspaces and all possible combinations of the unit subspaces.

With all the essential concepts associated with subspace defined, we are at the stage of defining concepts associated with time series discord mining.

Definition 6. (Time series discord). Given a time series T, the time series discord is the subsequence with maximum *dynamic time warping* (DTW) distance with its nearest neighbor, and the anomaly score is the DTW distance between the discord and its nearest neighbor.

In order to identify the discord, we search for the nearest neighbor of each subsequence based on the z-normalizing DTW distance within $\mathbf{T}$; then, store the distance between each subsequence and its nearest neighbor (see Fig. 2). Based on the stored distance values, we identify the discord from $\mathbf{T}$ in Fig. 2 by locating the subsequence with largest nearest neighbor distance.

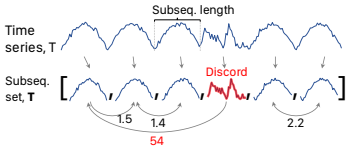

Fig. 2. The discord (red) is the subsequence with the largest distance with its nearest neighbor. Each subsequence's nearest neighbor is indicated by the curved arrow and the curved arrow is pointed *toward* its neighbor. The distances are shown next to the curved arrow.

4 System Architecture

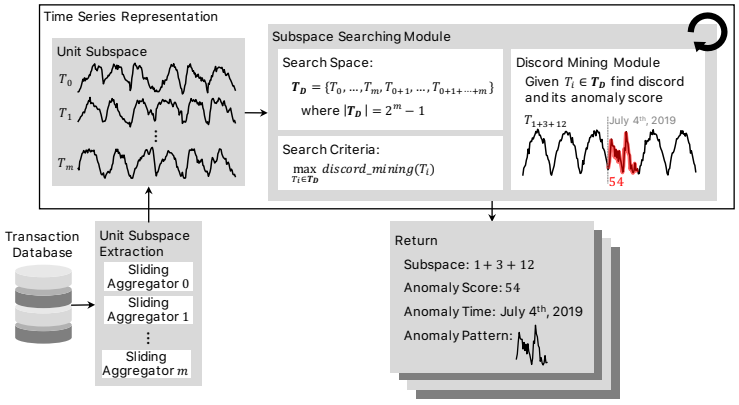

Fig. 3. The architecture of the proposed anomaly search system.

In Fig. 3, we provide an overview of the proposed anomaly retrieval system. First, a set of sliding aggregators are applied to the transaction database to extract unit subspaces.

We then feed the unit subspaces into the *Subspace Searching Module* (see Section 4.1). The *Subspace Searching Module* executes iteratively and searches for the subspace(s) with the largest possibility of being anomalous. The set of suggested subspace(s) is sent to *Discord Mining Module* for evaluating the anomaly score of the subspaces. The results of *Discord Mining Module* are then sent back to the *Subspace Searching Module* to guide the search direction in the next iteration. The goal of the *Subspace Searching Module* is to suggest the next anomalous subspace in each iteration and the goal of the *Discord Mining Module* is to evaluate the anomaly scores of the identified subspaces output by the *Subspace Searching Module*. Finally, once the iterative process is done (i.e., convergence is reached), a ranked list containing the identified anomalous subspaces, anomaly scores, temporal location of the anomalies, and the anomaly patterns are returned to the user for further investigation. The ranked list is formed by storing all the evaluated anomalies during the search process.

4.1 Subspace Searching Module

As we describe in Section 3, it is impossible to perform discord mining on all the combinations of unit subspaces for real-world transactional data because the size of search space is exponential with respect to the number of unit subspaces. We design two heuristic searching algorithms for the subspace search problem: greedy search and evolutionary search.

Greedy search: The greedy search method finds the most anomalous subspace by making a greedy choice at each iteration. We demonstrate the greedy search method with an example. Assume we have four unit subspaces (i.e., S_1, S_2, S_3, and S_4) initially. In the first step, we evaluate each unit subspace individually and find out that S_2 has the largest anomaly score. Next, we evaluate the combined subspace of S_2 with S_1, S_3, and S_4, separately. Let us say the combined subspace of S_2 and S_3 has the largest anomaly score; we aggregate them together ($S_2 + S_3 = S_{2,3}$). For the next step, we evaluate the aggregated spaces by combining $S_{2,3}$ with other unit spaces, i.e., S_1 and S_4, separately. The subspace produced by aggregating $S_{2,3}$ with S_1 has the largest anomaly score; therefore, we proceed with $S_{1,2,3}$ for the next iteration. Finally, we aggregate $S_{1,2,3}$ with the last unit subspace S_4 to form the last candidate subspace. By comparing the anomaly scores of S_2, $S_{2,3}$, $S_{1,2,3}$, and $S_{1,2,3,4}$, the algorithm returns a list of subspaces ordered based on anomaly scores.

Evolutionary Search: Evolutionary search [7, 14] is a heuristic optimization method. As its name suggests, evolutionary algorithms mimic the natural selection process, and applying such a method to our problem requires defining the following functions: genetic representation, fitness function, initialization, crossover, mutate, and selection strategy. We use a bit vector to represent a subspace, where the 1's in the vector indicating the presence of a unit subspace. The length of the bit vector is equal to the number of unit subspaces. For example, if we have unit subspaces S_1, S_2 and S_3, we use `[101]` to represent a particular subspace that is generated by combining S_1 and S_3. The fitness function is the anomaly score generated from the Discord Mining Module.

For the initialization, each individual within the population can be produced by generating a random binary vector. Given two parent binary vectors for the crossover, for every position of the offspring's binary vector, we randomly copy the value from one of the parents. If a mutation occurs, we randomly flip the mutated bit in the binary vector. For selection strategy, we use the tournament selection method with three tournament participants. The particular evolutionary algorithm we used is the $(\mu + \lambda)$ algorithm [9], and a ranked list of the individuals (subspaces) ordered based on anomaly scores is returned.

There are several hyperparameters in the evolutionary search method (i.e., population size per generation μ, number of offspring λ, probability of crossover p_{cx}, and number of generation n), and the time complexity is determined based on the hyperparameter setting. Specifically, the time complexity is $O(n\lambda)$ where n is the number of generations and λ is the number of offsprings. There are two major factors that users need to consider when deciding hyperparameter settings: the runtime requirement and the domain knowledge about the potential solution. Other model hyperparameters can be set based on the user's domain knowledge. For example, if users believe the solution should only consist of different unit subspaces, users should initialize the initial population with binary vectors with higher sparsity.

Greedy versus evolutionary search: There is no clear winner when comparing the greedy search method with the evolutionary search method. The decision on which method to adopt should be decided in a case-by-case fashion. To help the decision process, we show the cost comparison between the two approaches in Table 2.

The major differences between the two methods are 1) flexibility in computational cost and 2) the number of hyperparameters. The greedy search method has a fixed time complexity of $O(n^2)$ where n is the number of unit subspace. The time complexity is substantially better than the naive approach of $O(2^n)$, but it still could be too expensive for problems that demand a faster algorithm. Because the time complexity of the evolutionary search method only depends on the hyperparameter setting, the evolutionary search method can be parameterized in a way that the required speed can be achieved. The adjustable time complexity associated with evolutionary search comes with a disadvantage: it requires users to provide a set of hyperparameters while greedy search requires no hyperparameters. In other words, the greedy search method is much easier to use compared to the evolutionary algorithm. We perform an empirical comparison of the two methods in Section 5.2.

Table 2. Comparison between greedy and evolutionary search.

	Advantages	Disadvantages
Greedy	Fixed time complexity $O(n^2)$ No hyperparameter	Inflexible cost
Evolutionary	Flexible computational cost	Hyperparameters

4.2 Discord Mining Module

For the Discord Mining Module, we apply a nearest neighbor searching strategy based on DTW distances. The brute-force method computes the distances of all pairs of subsequences. The output is the subsequence with the largest DTW distance with the nearest neighbor. We call this method Discord Mining V0 (DM-V0).

Can we do better than DM-V0? Since computing the Euclidean distance is an order of magnitude faster than the DTW distance, it is possible to derive a faster discord mining algorithm using the fact that the z-normalized Euclidean distance for a pair of subsequences is always greater than their z-normalized DTW distance. We only need to evaluate the DTW distance of a subsequence pair when their Euclidean distance is greater than the current solution (i.e., the pair with the largest DTW distance in the current iteration of the search process); therefore, we can avoid unnecessary DTW distance computation using precomputed Euclidean distances. On top of that, we can also guide the search process using the precomputed Euclidean distances by evaluating the pair with the largest Euclidean distances first because the pair is more likely to contain the discord. We call the improved version Discord Mining V1 (DM-V1). The time complexity of DM-V1 is still $O(n^2m^2)$ where n is the number of subsequences and m is the length of a subsequence, but empirically it has a smaller runtime comparing to DM-V0 (see Fig. 4).

Can we do even better than DM-V1? Many existing time series data mining algorithms take advantage of the fact that subsequences could be overlapped with each other [22, 25, 40]. Their superior computational speed is achieved by avoiding redundant computation for the overlapped regions. It has been shown by Zhu et al. [40] that computing pairwise z-normalized Euclidean distance for every subsequence in a time series can be reduced from $O(n^2m)$ with naive implementation to $O(n^2)$ using a more optimal implementation. Using the $STOMP$ algorithm introduced in [40], we further improve the efficiency of DM-V1. Note, the original purpose of the $STOMP$ is to compute the matrix profile [36]. We modify the algorithm to return the pairwise distance matrix (i.e., all computed distance profiles [36]) instead of the matrix profile. We call the newly-introduced algorithm, Discord Mining V2 (DM-V2).

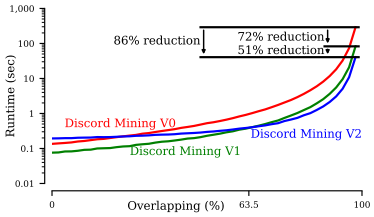

Fig. 4. The runtime of DM-V0, DM-V1, and DM-V2 under different overlapping condition. Note, the y-axis is runtime on a logarithmic scale.

When do we use DM-V2 versus V1? DM-V2 takes advantage of the fact that subsequences within the subsequence set are overlapped with each other. However, the runtime of the $STOMP$ algorithm is longer than pair-wise Euclidean distance computation when the overlap between subsequences is small. To study the relationship between the runtime of different discord mining methods and the overlap ratio, we have performed experiments on a synthetic random walk time series ($|T| = 2,880$). We set the subsequence length to 48, and the overlap between consecutive subsequences are varied from 0 to 47. The experiment is repeated 100 times and the average value is reported; the result is presented in Fig. 4.

DM-V1 is faster than DM-V0 which shows how precomputing the z-normalized Euclidean distance can indeed reduce the search time. When the overlap is close to

100% of the subsequence length, the runtime reduction is 72% by replacing DM-V0 with DM-V1. When DM-V2 is adopted, the runtime reduction is 86% comparing to DM-V0, and the runtime reduction is 51% comparing to DM-V1. On the contrary, when the subsequences have zero overlaps, the runtime of DM-V1 is the shortest compared with the alternatives, and the runtime for DM-V2 requires an extra 119 milliseconds compared to DM-V1 and 58 milliseconds compared to DM-V0. The runtime for DM-V1 and DM-V2 intersect when the overlap of subsequence is around 63.5%. In other words, to achieve the optimal performance under this particular experiment setup, we should use DM-V1 when the overlap between subsequence is less than or equal to 63.5% and DM-V2 when the overlap is greater than 63.5%.

4.3 Discussion

The goal of our anomaly retrieval system is to obtain a list of anomalies ordered based on their corresponding anomaly scores, and the anomaly with the highest anomaly score may not be a true anomaly from the user's standpoint. However, by providing users a list of anomaly candidates with its contextual information (i.e., which subspace the candidate is from, the candidate's temporal location, the shape of the anomaly pattern), the user can investigate further using the contextual information of each candidate. Such interaction is similar to how people use an online search engine.

Our two-module design can be easily extended for different applications. Currently, we use a nearest neighbor-based method to mine time series discord since we find it is suitable for finding anomalies (e.g., extreme values, abnormal trends, and sudden changes) in transactional time series data. In other applications, the Discord Mining Module can be replaced by a more suitable anomaly mining method for the application. As a result, we fix the Discord Mining Module design to the time series discord-based method and focus on comparing different subspace searching methods in Section 5.

5 Evaluation

The experiments are all conducted on a Linux server with Intel Xeon CPU E5-2650 v4. We present several alternative algorithms for subspace searching to compare with our system firstly. We then perform a stress test with synthetic transactional data to understand our system's runtime and output quality under different scenarios. Finally, we evaluate our anomaly retrieval system on real transactional data to show its effectiveness in real-world scenarios.

5.1 Alternative Approaches

Now, we introduce several alternative algorithms for the Subspace Searching Module to compare with greedy and evolutionary search.

- **All-dimension**: The method returns the subspace that consists of all dimensions, i.e., it aggregates all the unit subspaces then returns it as the output.

– **One-best dimension**: The method computes the anomaly score associated with each dimension (i.e., unit subspace), then returns the dimension (i.e., unit subspace) with the largest anomaly score.
– **Hierarchical clustering**: The method performs hierarchical clustering on dimensions, which means that dimensions with similar time series are clustered together. Suppose we have n dimensions, the clustering process groups the dimensions in $n-1$ steps. In each step, the hierarchical clustering algorithm merges either a dimension or a pre-existing cluster (i.e., grouped dimensions) with another dimension or pre-existing cluster. We consider each cluster of dimensions as a subspace, and we evaluate each subspace then return the ranked list of subspaces ordered by the anomaly scores. This method explores the possibility of using the similarity among dimensions for reducing the search space.

5.2 Synthetic Data

Methodology: Since it is beyond the modern computers' capability to obtain the optimal solution on real-world transactional data, we generate a set of synthetic data where the set of all possible combinations of unit subspaces is small enough for the brute-force subspace searching. Additionally, we generate the time series for each unit subspace directly instead of raw transactional data to streamline the data generation process as both the Subspace Searching Module and the Discord Mining Module work on time series representation of a transaction database. The synthetic data generated using the default experiment setting consists of 8 unit subspaces, and each unit subspace consists of 30 days of transactional data where each day is represented with 48 data points.

To study the effect of various variables associated with the synthetic dataset on the runtime and quality of the solution, we have varied one variable in the default experiment setting for each set of experiments. We generate 100 synthetic data points using different random seeds for each experiment setting to minimize the randomness effect.

To obtain the optimal solution, we generate all the subspaces $\mathbf{S}_{bf}$ (i.e., all combinations of the unit subspaces) by brute force. Fig. 5 shows the most anomalous subspace and the least anomalous subspace. The anomalies are highlighted in red.

Fig. 5. The time series data of the most and least anomalous subspace.

Aside from the runtime, we also measure the performance of our system with *averaged rank* which captures the quality of the approximated solution. The averaged rank is computed as follows: given a subspace S discovered by an approximated algorithm, we find the rank of S in $\mathbf{S}_{bf}$. Because we repeat the experiment on 100 synthetic data points generated using different random seeds (with the same experiment setting), we compute the average of these 100 ranks and report the averaged rank as the performance measurement of the solution quality.

In addition to the two subspace search approaches, we also include the result of a random baseline. The random baseline returns a random subspace as the solution. Out of all the alternative approaches, we only include the hierarchical clustering-based

method because all the other approaches are only capable of returning subspaces that consist of either one unit subspace or all unit subspaces. For all the experiments, we use multi-thread implementation with the number of threads set to 48. We use DM-V1 in all experiments as there is no overlap between subsequences.

Results: The experiment result is summarized in Fig. 6. The first row of Fig. 6 shows the averaged rank (i.e., quality of solution) under different Subspace Searching Modules. The number of unit spaces ranges from 4 to 16, constituting $2^4 - 1$ to $2^{16} - 1$ subspaces. As the number of unit subspace increases exponentially, the ranks of all the subspace searching methods increase correspondingly. All three subspace searching methods grow at a much slower rate compared to the exponential growth of the random baseline. We also observe that the performance of the proposed methods (i.e., greedy and evolutionary) is better than the hierarchical clustering-based method. On the contrary, the variable of the number of days does not have any correlation with the average rank: increasing or decreasing the number of days does not change the search space for anomaly retrieval. When the number of samples per day increases, the average ranks of the proposed algorithms slightly improve.

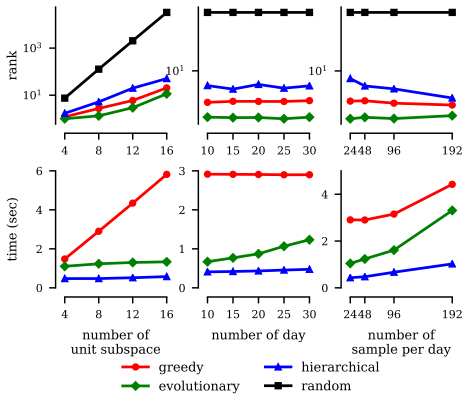

Fig. 6. Experiment result on synthetic data. Note that the y-axis of the plot in the first row is on a logarithmic scale.

The second row of Fig. 6 depicts the runtime under different Subspace Searching Modules. We do not include the runtime of the random baseline in this study because there is no subspace search operation in the random baseline. When we change the number of unit subspaces, the size of the search space explored by the Subspace Searching Module is changed, thus varying such a variable could influence the runtime of the Subspace Searching Module. For the greedy method, the result shows that the computational cost is linear to the number of unit subspaces. For the evolutionary method, the time complexity only depends on the hyperparameter settings of the algorithm. Since we use the same hyperparameter setting (i.e., the default setting presented in the next section) throughout the experiment, the runtime is not affected by the change in the number of unit subspaces. Note that though the runtime of the greedy method more than triples when the number of subspaces grows from 4 to 16, it is still much faster than the brute force method as the brute force method requires a whopping 33 minutes to find the solution compared to the six seconds for the greedy method.

When we change either the number of days or the number of samples per day, the module that is mostly affected by the change is the Discord Mining Module. Increasing either of these variables, in theory, should increase the computational time quadratically

of the Discord Mining Module. Nevertheless, the number of days has a limited effect on the greedy search method and the growing trend of the evolutionary search method suggests a quadratic growth rate. The number of samples per day has a similar impact on the runtime of both the greedy search method and the evolutionary search method. The trend suggests a quadratic growth in both methods' runtime with respect to the number of samples per day. Overall, the hierarchical clustering-based method is faster than the other methods due to its smaller search space; however, it also has the worst averaged rank for the same reason. We also generated figures with other common performance measurements for retrieval systems (i.e., MAP and NDCG [28]). Because the conclusion remains the same, we omit those figures for brevity.

Sensitivity Analysis for Evolutionary Search: As there are many hyperparameters associated with the evolutionary search method, we perform sensitivity analysis using synthetic data with 12 unit subspaces, 30 days of transactional data, and 48 data points per day. This analysis also compares greedy search with evolutionary search under different hyperparameter settings to help users decide on which subspace searching method to use. The result is shown in Fig. 7. Similar to Fig. 6, the y-axis indicates performance measurements like rank and time while the x-axis indicates varied hyperparameters. The "default" hyperparameter setting is: $\mu = 64$, $\lambda = 32$, $p_{cx} = 0.7$, and $n = 8$ where μ is population size per generation, λ is number of offspring, p_{cx} is probability of crossover, and n is number of generation.

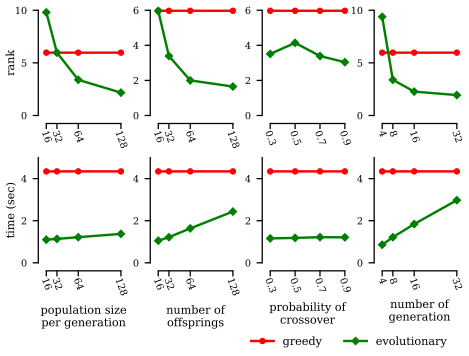

Fig. 7. Performance of evolutionary search under different hyperparameter settings.

First of all, the runtime mostly depends on λ and n. When we increase either λ or n, the runtime also increases linearly and the rank improves considerably before saturation. The value associated with μ also has a positive correlation with the runtime, but the growth rate is much less than λ and n while the improvement in rank remains prominent. On the contrary, p_{cx} does not affect the runtime, and setting it to 0.9 gives us the best result comparing to other p_{cx} settings. Note, although the evolutionary search method almost always outperforms the greedy search throughout the sensitivity analysis, there are still cases where the evolutionary search method is surpassed by the greedy search method when the hyperparameter setting is not ideal. This demonstrates the benefit of the greedy search method: there is no sensitivity analysis required for using greedy search because there are no hyperparameters associated with the greedy search method. Similar to previous experiments, the figures with other performance measurements are omitted for brevity.

5.3 Real-world Transactional Data

Data Collection: We collect all the California state transactional data in July 2018 from a payment company. Each transaction is associated with tens of attributes. Particularly, we determine that the unit subspaces are defined by Merchant Category Code (MCC). The MCC is the code used to determine the business type (e.g., department store, clothing store, and grocery) of a merchant. When transactions of different MCC are analyzed together (i.e., combining their corresponding unit subspace to form a subspace), it could reveal valuable information regarding certain business sectors. Discovered anomalous MCC subspaces can be interpreted and used as guidance for designing business strategies. For example, let's say MCC_1 stands for department stores, and MCC_2 stands for clothing stores. If an anomalous event with unusual rising transaction volume is detected in the subspace (MCC_1, MCC_2) on a specific date, such an event could indicate that there could be a big sale on the date for stores that sell garments.

The dataset is 70GB and consists of over 600 million transactions from 415 MCCs. In each trial of the experiment, we randomly select one day and a subset of MCCs (1% of the MCCs) in the transaction database; we then randomly add synthesized transactions belonging to the selected MCCs to the day and randomly remove transactions from the selected MCCs occurring on that day. We repeat the experiment 16 times. When we apply the sliding aggregator, we use a sliding window of a half-hour and a hop size of the same length. The particular statistics we compute with the sliding aggregator is the sum of the transaction amounts spent in the window.

Since we have the ground truth about the temporal location of the anomalies, we return the averaged rank of the *real* anomalous day in the ranked list returned by different anomaly retrieval systems over the 16 trials. To obtain the ranked list, the anomaly score associated with each day in the subspace returned by each system is computed; then the computed scores are used as the sorting criteria. Because the tested systems are retrieval systems, we measure the performance of the systems using information retrieval performance measures like MAP [28] and NDCG [28] to evaluate the quality of the ranked list in addition to the averaged rank. Note, for the averaged rank, a lower number means better performance; for MAP and NDCG, a higher number means better performance.

Table 3. Experiment results with real transactional data.

Method	Average Rank	MAP	NDCG	Runtime (Sec)
Random	15.50	0.63	0.34	-
All-dimension	20.75	0.36	0.24	0.02
One-best Dimension	17.31	0.47	0.28	9.58
Hierarchical Clustering	17.31	0.47	0.28	4.17
Greedy Search	**5.94**	**0.86**	**0.46**	1,345.50
Evolutionary Search	8.63	0.72	0.39	87.62

Results: The experiment results are shown in Table 3. All alternative approaches listed in Section 5.1 are examined. The proposed system, either with greedy search or evolutionary search, considerably outperforms the baseline methods with the greedy search being slightly better than the evolutionary search. On the contrary, either the one dimension or all dimension system fails to reliably detect the injected anomalies as their corresponding performance is even worse than the random baseline. The use of all dimension system fails because the injected anomalies only affect 1% of the unit subspace. Additionally, the one dimension system fails because it cannot locate the

anomaly when only one dimension is considered. The anomaly on one dimension is too small to be captured. The hierarchical clustering-based system also produces poor results as the assumption that the anomalous space consists of similar unit subspaces does not hold for our database. In terms of the runtime, the proposed system, even with the slower greedy search method, is capable of running in real-time as the runtimes are all less than the data sampling period (i.e., 1,800 sec.). Similar to the results we present in Fig. 6, the evolutionary search method is capable of finding a solution that has a comparable quality with the solution located by greedy search with a shorter runtime.

6 Conclusion

In this paper, we propose an anomaly retrieval system for high-dimensional time series. The proposed system consists of two integrated modules, i.e., subspace searching module and discord mining module. We implement the proposed system and perform a comprehensive evaluation with synthetic data and real-world transactional data. Our experimental results show that our system outperforms the baseline algorithms with an execution time suitable for real-time analysis in the production environment. It only takes 22 minutes to process one month of transaction records (i.e., 600 million records) with the greedy search variant of the proposed system, and 1.5 minutes for the evolutionary search variant.

References

1. Aggarwal, C.C., Yu, P.S.: Outlier detection for high dimensional data. In: ACM Sigmod Record (2001)
2. Breunig, M.M., Kriegel, H.P., Ng, R.T., Sander, J.: Lof: identifying density-based local outliers. In: ACM sigmod record (2000)
3. Chakraborty, K., Mehrotra, K., Mohan, C.K., Ranka, S.: Forecasting the behavior of multivariate time series using neural networks. Neural networks (1992)
4. Chandola et al.: Anomaly detection: A survey. ACM computing surveys (2009)
5. Dean, D.J., Nguyen, H., Gu, X.: Ubl: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems. In: ICAC (2012)
6. Duan et al.: Mining outlying aspects on numeric data. DMKD (2015)
7. Eiben et al.: Introduction to evolutionary computing. Springer (2003)
8. Faruk, D.Ö.: A hybrid neural network and arima model for water quality time series prediction. Engineering Applications of Artificial Intelligence (2010)
9. Fortin et al.: Deap: Evolutionary algorithms made easy. JMLR (2012)
10. Gong, S., Zhang, Y., Yu, G.: Clustering stream data by exploring the evolution of density mountain. VLDB (2017)
11. Gupta et al.: Outlier detection for temporal data: A survey. IEEE TMKD (2013)
12. Han, J., Pei, J., Kamber, M.: Data mining: concepts and techniques. Elsevier (2011)
13. He et al.: Tscope: Automatic timeout bug identification for server systems. In: ICAC (2018)
14. Holland, J.H.: Genetic algorithms. Scientific american (1992)
15. Kaastra, I., Boyd, M.: Designing a neural network for forecasting financial and economic time series. Neurocomputing (1996)
16. Keller, F., Muller, E., Bohm, K.: Hics: High contrast subspaces for density-based outlier ranking. In: ICDE (2012)

17. Keogh, E., Lin, J., Lee, S.H., Van Herle, H.: Finding the most unusual time series subsequence: algorithms and applications. KIS (2007)
18. Kriegel, H.P., Kröger, P., Schubert, E., Zimek, A.: Outlier detection in axis-parallel subspaces of high dimensional data. In: PAKDD (2009)
19. Kriegel, H.P., Schubert, M., Zimek, A.: Angle-based outlier detection in high-dimensional data. In: SIGKDD (2008)
20. Malhotra, P., Vig, L., Shroff, G., Agarwal, P.: Long short term memory networks for anomaly detection in time series. In: Proceedings. Presses universitaires de Louvain (2015)
21. Manzoor, E., Lamba, H., Akoglu, L.: xstream: Outlier detection in feature-evolving data streams. In: SIGKDD (2018)
22. Mueen et al.: Time series join on subsequence correlation. In: ICDM (2014)
23. Müller, E., Schiffer, M., Seidl, T.: Statistical selection of relevant subspace projections for outlier ranking. In: ICDE (2011)
24. Na, G.S., Kim, D., Yu, H.: Dilof: Effective and memory efficient local outlier detection in data streams. In: SIGKDD (2018)
25. Rakthanmanon et al.: Searching and mining trillions of time series subsequences under dynamic time warping. In: SIGKDD. ACM (2012)
26. Salehi, M., Leckie, C., Bezdek, J.C., Vaithianathan, T., Zhang, X.: Fast memory efficient local outlier detection in data streams. TKDE (2016)
27. Santora, M.: In hours, thieves took $45 million in a.t.m. scheme. https://www.nytimes.com/2013/05/10/nyregion/eight-charged-in-45-million-global-cyber-bank-thefts.html (2013)
28. Schütze, H., Manning, C.D., Raghavan, P.: Introduction to information retrieval, vol. 39. Cambridge University Press Cambridge (2008)
29. Siffer, A., Fouque, P.A., Termier, A., Largouet, C.: Anomaly detection in streams with extreme value theory. In: SIGKDD. ACM (2017)
30. Su, Y., Zhao, Y., Niu, C., Liu, R., Sun, W., Pei, D.: Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In: SIGKDD (2019)
31. Vinh, N.X., Chan, J., Romano, S., Bailey, J., Leckie, C., Ramamohanarao, K., Pei, J.: Discovering outlying aspects in large datasets. DMKD (2016)
32. World Bank Group: World bank open data. https://data.worldbank.org/ (2019)
33. Wu et al.: Promotion analysis in multi-dimensional space. VLDB (2009)
34. Ye, M., Li, X., Orlowska, M.E.: Projected outlier detection in high-dimensional mixed-attributes data set. Expert Systems with Applications (2009)
35. Yeh, C.C.M., Kavantzas, N., Keogh, E.: Matrix profile vi: Meaningful multidimensional motif discovery. In: ICDM (2017)
36. Yeh et al.: Matrix profile i: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In: ICDM (2016)
37. Zhang, J., Gao, Q., Wang, H.: Spot: A system for detecting projected outliers from high-dimensional data streams. In: ICDE (2008)
38. Zhang, L., Lin, J., Karim, R.: An angle-based subspace anomaly detection approach to high-dimensional data: With an application to industrial fault detection. Reliability Engineering & System Safety (2015)
39. Zhang, L., Lin, J., Karim, R.: Sliding window-based fault detection from high-dimensional data streams. IEEE Transactions on Systems, Man, and Cybernetics: Systems (2016)
40. Zhu et al.: Matrix profile ii: Exploiting a novel algorithm and gpus to break the one hundred million barrier for time series motifs and joins. In: ICDM (2016)