The VLDB Journal (2024) 33:481–505
https://doi.org/10.1007/s00778-023-00815-y

REGULAR PAPER

Efficient detection of multivariate correlations with different correlation measures

Jens E. d'Hondt[1] · Koen Minartz[1] · Odysseas Papapetrou[1]

Received: 20 March 2023 / Revised: 22 August 2023 / Accepted: 14 September 2023 / Published online: 11 October 2023

Abstract
Correlation analysis is an invaluable tool in many domains, for better understanding the data and extracting salient insights. Most works to date focus on detecting high pairwise correlations. A generalization of this problem with known applications but no known efficient solutions involves the discovery of strong multivariate correlations, i.e., finding vectors (typically in the order of 3–5 vectors) that exhibit a strong dependence when considered altogether. In this work, we propose algorithms for detecting multivariate correlations in static and streaming data. Our algorithms, which rely on novel theoretical results, support four different correlation measures, and allow for additional constraints. Our extensive experimental evaluation examines the properties of our solution and demonstrates that our algorithms outperform the state-of-the-art, typically by an order of magnitude.

Keywords Similarity search · Multivariate correlations · Time series · Streaming data

1 Introduction

Correlation analysis is one of the key tools in the arsenal of data analysts for understanding the data and extracting insights. For example, in neuroscience, a strong correlation between activity levels in two regions of the brain indicates that these regions are strongly interconnected [20]. In finance, correlation plays a crucial role in finding portfolios of assets that are on the Pareto-optimal frontier of risk and expected returns [30], and in genetics, correlations help scientists detect cause factors for potentially hereditary syndromes.[1] In databases, similarity measure like correlations are occasionally used in theta joins to allow for softer joining conditions than pure object equality [21]. Furthermore, when treated as a generalization of functional dependencies, correlations are also used for optimizing access paths in databases [47].

Multivariate correlations, also known as high-order correlations, extend the concept of pairwise correlations to relationships among three or more variables. These variables may represent various forms of data, such as time series or other high-dimensional data stored as vectors.[2] Multivariate correlations should not be confused with pairwise correlations of multivariate time series. The former refers to correlations involving three or more distinct variables/vectors, whereas the latter deals with correlations of only two multivariate time series. In the last few years, multivariate correlations found extensive use in diverse domains. Detection of ternary correlations in fMRI time series improved the understanding of how different brain regions work in cohort for executing different tasks [2, 3]. For instance, the activity of the left middle frontal region was found to have a high correlation with the total activity of the right superior frontal and left inferior frontal regions while the brain was processing audiovisual stimulus. This insight suggests that the left middle frontal has an integrative role of assimilating information from the other two regions, which was not possible

[1] A prime example is the Spark project for discovering gene properties related to the manifestation of the autism spectrum disorder, which led to a list of genes and their correlated symptoms [17].

[2] Although we will mostly refer to the more general case of vectors in this paper, the data often consists of time series—possibly with live updates.

✉ Jens E. d'Hondt
j.e.d.hondt@tue.nl

Koen Minartz
k.minartz@tue.nl

Odysseas Papapetrou
o.papapetrou@tue.nl

[1] Eindhoven University of Technology, De Zaale 1, 5600 MB Eindhoven, The Netherlands

to find by looking only at pairwise correlations. In climate science, a ternary correlation led to the characterization of a new weather phenomenon and to improved climate models [29]. In machine learning, multivariate information-theoretic measures have increasingly served as learning objectives or regularizers for training of neural networks aimed at optimizing the correlation among multiple variables. Usage of such regularizers lead to improved robustness, generalizability, and interpretability of the models [4, 7, 8]. It is also stipulated that a more thorough look at multivariate correlations will open doors in the fields of genomics [6, 52] and medicine [28, 32].

Accordingly, several measures and algorithms for discovering strong multivariate correlations have been proposed, such as Tripoles [2], Multipoles [3], Canonical Correlation Analysis (CCA) [23], and Total Correlation (TC) [35, 36, 46, 52]. However, the proposed algorithms do not sufficiently address the fundamental impediment on the discovery of strong multivariate correlations, which is the vast search space—all combinations of vectors that need to be examined. Unfortunately, apriori-like pruning techniques do not apply for the general case of multivariate correlations. For example, consider the three time series from finance, presented in Fig. 1. In this example, the pairwise correlation between all pairs of the three time series is comparatively low, whereas the time series created by averaging QAN and RDF is strongly correlated to MCP.[3] Therefore, a correlation value of any pair of vectors does not provide sufficient information as of whether these vectors may participate together in a ternary (or higher-order) correlation. Simultaneously, an exhaustive algorithm that iterates over all possible combinations implies combinatorial complexity, and cannot scale to reasonably large datasets. Indicatively, in a small data set of 100 vectors, detection of all ternary high correlations requires iterating over 1 million candidates, whereas finding quaternary high correlations on 1000 vectors involves 1 trillion combinations. The mere generation and enumeration of these combinations already becomes challenging. Therefore, smart algorithms are needed to drastically reduce the search space and computational complexity.

Existing algorithms follow at least one of the following approaches: (a) they consider constraining definitions of multivariate correlations that enable apriori-like filtering [3, 35, 52], (b) they rely on hand-crafted assumptions of the user query, which may be too constraining for other application scenarios [2, 3, 52], or, (c) they offer approximate results, with no guarantees [2, 3]. Even though these algorithms are very useful for their particular use cases, they are not generally applicable.

[3] Weighted averages of stock prices are commonly considered in risk management to evaluate portfolio performance, diversity, and volatility [38].

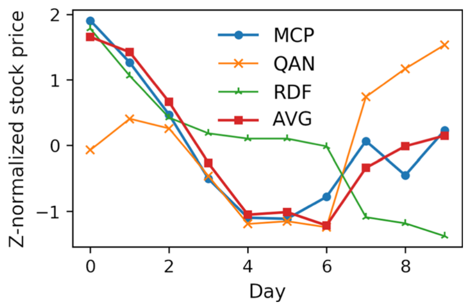


Fig. 1 Normalized daily closing prices for stocks traded at the Australian Securities Exchange

In this work, we follow a more general direction. First, we also consider correlation measures that are not suitable for apriori-like pruning. Second, in contrast to some of the earlier work, we abide by Ockham's razor: we prioritise discovery of the less complex multivariate correlations—the ones that contain the smallest number of vectors. We opt for this approach since correlations between a few variables are more intuitive and interpretable than their counterparts with many variables. Third, we consider different algorithmic variants: an exact threshold variant that returns all correlations higher than a threshold τ, and an exact top-κ variant that returns the top-κ highest correlations. We also discuss the case of progressively finding results, and extend the proposed algorithms to a dynamic context, for handling streaming updates.

We evaluate our algorithms on 7 datasets and compare them to the state-of-the-art. Our evaluation demonstrates that we outperform the existing methods, frequently by several orders of magnitude. Finally, we show that the progressive version of the algorithm produces around 90% of the answers in 10% of the time.

The remainder of the paper is structured as follows. In the next section, we formalize the problem and discuss the preliminaries and related work. We then propose the algorithmic variants for the case of static data (Sect. 3), and the streaming extension of the algorithm (Sect. 4). Section 5 summarizes the experimental results. We conclude in Sect. 6.

2 Preliminaries

We start with a discussion of the multivariate correlation measures that we will be considering in this work. We then formalize the problem and discuss prior work on similar multivariate correlation measures.

2.1 Correlation measures

Our work focuses on both types of multivariate correlation measures: **a** bivariate correlations *over aggregated vectors* (two-sided), and **b** specialized multivariate measures (one-sided).

Bivariate correlations over aggregates. Given two sets of vectors X and Y, a bivariate correlation over aggregated vectors is defined as

$$Corr(X, Y) = Corr(\text{Agg}(X), \text{Agg}(Y)) \quad (1)$$

with *Corr* being a bivariate correlation function such as Pearson Correlation, and $Agg(X)$ being a linear combination of the vectors in X. In this work, we consider element-wise averaging combined with Pearson Correlation and Euclidean Similarity [42], referred to as *PC* and *ES*, respectively. Pearson Correlation is defined as $\rho(x, y) = \frac{\text{cov}(x,y)}{\sigma_x \sigma_y}$ with σ_x denoting the standard deviation of some vector x, and is a widely used measure for measuring the linear dependence between two variables. Euclidean Similarity is defined as $ES(x, y) = \frac{1}{1+d(x,y)}$ with $d(\cdot, \cdot)$ denoting the Euclidean distance, and is extensively used for k-nearest neighbors queries and range queries [13, 15].

Multipole. The multipole correlation $MP(X)$ measures the linear dependence of an input set of vectors X [3]. Specifically, let $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_n$ denote n z-normalized input (column) vectors, and $\mathbf{X} = [\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_n]$ the matrix formed by concatenating the vectors. Then:

$$MP(X) = 1 - \min_{\mathbf{v} \in \mathbb{R}^n, \|\mathbf{v}\|_2 = 1} \text{var}(\mathbf{X} \cdot \mathbf{v}^T) \quad (2)$$

The value of $MP(X)$ lies between 0 and 1. The measure takes its maximum value when there exists perfect linear dependence, meaning that there exists a vector $\mathbf{v}$ with norm 1, such that $\text{var}(\mathbf{X} \cdot \mathbf{v}^T) = 0$. Notice that multipoles is not equivalent to, nor a generalization of *PC* or *ES*. By definition, *MP* assumes optimal weights (vector $\mathbf{v}$ is such that the variance is minimized), whereas for *PC* and *ES*, the aggregation function for the vectors (e.g., averaging) is determined at the definition of the measure. Furthermore, $MP(\cdot)$ expresses the degree of linear dependence within a single set of vectors, whereas for bivariate measures, two distinct, non-overlapping vector sets are considered.

Total correlation. Total correlation $TC(X)$ (also known as multi-information [43] or multivariate constraint [18]) is a generalization of the (pairwise) mutual information measure. It measures the redundancy or dependence among a set of n random variables $X = \{X_1, \ldots, X_n\}$ as the KL-divergence from the joint distribution $p(X_1, \ldots, X_n)$ to the product of the marginal distributions $p(X_1) \ldots p(X_n)$ [46]. This can be reduced to the difference of entropies:

$$TC(X) = \sum_{i=1}^{n} H(X_i) - H(X_1, \ldots, X_n) \quad (3)$$

with $H(X_i)$ denoting Shannon's entropy of $X_i \in X$.

2.2 Problem definition

Consider a set $\mathcal{V} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots \mathbf{v}_n\}$ of d-dimensional vectors, and a multivariate correlation measure *Corr*, both provided by the data analyst. Function *Corr* accepts either one or two vector sets (subsets of $\mathcal{V}$) as input parameters, and returns a scalar. Hereafter, we will be denoting the correlation function with $Corr(X, Y)$, with the understanding that for the definitions of *Corr* that expect one input (i.e., *MP* and *TC*), Y will be empty. We consider two query types:

Query 1: Threshold query: For a user-chosen correlation function *Corr*, correlation threshold τ, and parameters $p_l, p_r \in \mathbb{N}$, find all pairs of sets $(X \subset \mathcal{V}, Y \subset \mathcal{V})$, for which $Corr(X, Y) \geq \tau$, $X \cap Y = \emptyset$, $|X| \leq p_l$ and $|Y| \leq p_r$.

Query 2: Top-κ query: For a user-chosen correlation function *Corr*, and parameters κ, p_l, $p_r \in \mathbb{N}$, find the κ pairs of sets $(X \subset \mathcal{V}, Y \subset \mathcal{V})$ that have the highest values $Corr(X, Y)$, such that $X \cap Y = \emptyset$, $|X| \leq p_l$, and $|Y| \leq p_r$.

The combination of p_l and p_r controls the desired complexity of the answers. Smaller $p_l + p_r$ values yield results that are easier to interpret, and arguably more useful to the data analyst.

Complementary to the two query types, users may also want to specify additional constraints, relating to the targeted diversity and significance of the answers. We consider two different constraints, but other constraints (e.g., the weak-correlated feature subset constraint of [52]) can also be integrated in the algorithm into a similar manner:

Irreducibility constraint: For each (X, Y) in the result set, there exists no (X', Y') in the result set such that $X' \subseteq X$, $Y' \subseteq Y$, and $\left(X', Y'\right) \neq (X, Y)$. Intuitively, if $Corr(X', Y') \geq \tau$, then no supersets of X' and Y' should be considered together. This constraint prioritizes simpler answers.

Minimum jump constraint: For each (X, Y) in the result set, there exists no (X', Y') such that $X' \subseteq X$, $Y' \subseteq Y$, $\left(X', Y'\right) \neq (X, Y)$, and $Corr(X, Y) - Corr(X', Y') < \delta$. This constraint, which was first proposed in [2], discards solutions where a vector in $X \cup Y$ contributes less than δ to the increase of the correlation.

For top-κ queries, these constraints are ill-defined. For example, consider the irreducibility constraint, and assume $Corr(X, Y) = 0.9$, and $Corr(X', Y') = 0.8$, where $X' \subset X$ and $Y' \subset Y$. In this case, the definition of top-κ does not dictate which of (X, Y) or (X', Y') should be in the answer set.

For conciseness, we will use $Corr(p_l)$ and $Corr(p_l, p_r)$ to denote the combination of the correlation measure, and the user-chosen values of p_l and p_r. For example, $PC(2, 1)$ will identify the combinations of sets of vectors of size 2 and 1 with high Pearson correlation, whereas pattern $MP(4)$ will identify the combinations of 4 vectors with high multipole correlation.

2.3 Related work

Several algorithms exist for efficiently finding highly correlated pairs in large data sets of high-dimensional vectors, e.g., time series. For example, StatStream [53] and Mueen et al. [34] both map pairwise correlations to Euclidean distances. They then exploit Discrete Fourier Transforms, grid-based indexing and dynamic programming to reduce the search space. Other works also enable indexing of high-dimensional vectors in the Euclidean space [11, 40]. However, these works are not applicable for multivariate correlations, since two vectors may have a low pairwise correlation with a third vector, whereas their aggregate may have a high correlation (see, e.g., the example of Fig. 1). Prior work addressing multivariate correlations propose algorithms that rely on additional constraints for their pruning power. Agrawal et al. investigate the problem of finding highly-correlated tripoles [2]. Tripoles is a special case of the *PC* measure, where $|X| = 2$ and $|Y| = 1$ (i.e., $PC(2, 1)$). Their algorithm, named CoMEt, relies on the minimum jump constraint for effective pruning. Compared to tripoles, our work handles the more general definition of Pearson correlation over aggregated vectors, allowing more vectors on the left- and right-hand side. Moreover, our work relies on novel theoretical results to prune the search space and can scale to larger datasets regardless of the introduction of any additional constraints (e.g., minimum jump or irreducibility).

Algorithms for discovering high correlations according to the Multipole measure (Eq. 2) were first proposed in [3], with the introduction of the CoMEtExtended algorithm. Both CoMEt and CoMEtExtended are approximate and rely on clique enumeration to efficiently explore the search space. Their efficiency depends on a parameter ρ that trades off result completeness for performance. The minimum jump constraint also becomes relevant to reduce computational effort. For settings of ρ that result in reasonable computation times, the two algorithms yield a substantially more complete result set compared to methods like l_1—regularization and structure learning-based techniques. Still, the two algorithms do not come with completeness or accuracy guarantees. In contrast, our work is exact—it always retrieves all answers—and outperforms both algorithms.

With respect to Total Correlation, Nguyen et al. [35] propose an algorithm for groups of columns in a database with high Total Correlation. The method analyzes patterns in pairwise correlations (i.e., mutual-information) to identify quasi-cliques of highly correlated column groups, and compute lower bounds on their total correlation. However, it misses strongly correlated groups with low pairwise correlations, which are arguably the most interesting cases. As such, the method is effectively an approximation algorithm.

In another work, Zhang et al. developed an algorithm that discovers sets of binary vectors with a high total correlation value [52]. However, the method is again approximate, limited to data with binary features only, and relies on a limiting weak-correlated subset constraint. In contrast, our work returns a guaranteed complete set of results and works on all major data types.

In the supervised learning context, subset regression appears to be closely related to multivariate correlation mining. The goal of this feature selection problem is to select the best p predictors out of n candidate features [10]. Our problem differs from the above in that we aim to find interesting patterns in the data, rather than finding the best predictors for a *given* dependent variable. Furthermore, instead of finding only the highest correlated vector set, our goal is to find a *diverse set* of results as we argue that that will help domain expert assess the results more on qualitative aspects, gaining more insights.

Another similar problem is that of similarity search on multivariate time series [49, 50]. Here, the goal is to find *all pairs of multivariate time series* (e.g., weather sensors measuring both temperature and wind speed) with a high similarity value, based on some specialized measure such as the PCA similarity factor [45], or the extended Frobenius norm [48]. Effectively, this extends classic similarity search by adding a degree of freedom (DoF) in the number of variables *per time series*, increasing the search space cardinality from $O(n^2)$ to $O((pn)^2)$ for p-variate time series. In contrast, our problem extends classic similarity search by adding a DoF in the number of time series *per combination*, growing the search space to $O(n^p)$. Although this problem seems similar, its challenges differ significantly from similarity search on multivariate time series and can lead to different results and insights.

Table 1 summarizes the properties of the most closely related work out of the discussed ones.

3 Detection of multivariate correlations in static data

The main challenge in detecting strongly correlated vector sets stems from the combinatorial explosion of the number of candidates that need to be examined. In a dataset of n vectors, there exist at least $O\left(\sum_{p=2}^{p_l+p_r}\binom{n}{p}\right)$ possible combinations for a correlation pattern $Corr\ (p_l, p_r)$. Even if each possible combination can be checked in constant time, the enumeration of all combinations still requires significant computational effort.

Table 1 Comparison to the most relevant related work for multivariate correlations

	Completeness	Require constraints	Correlation measures	Query types	Data formats
[2]	No	Yes	$PC(1, 2)$	Threshold	Static
[3]	No	Yes	$MP(\cdot)$	Threshold	Static
[35]	No	No	$TC(\cdot)$	Threshold	Static
[52]	Yes	Yes	$TC(\cdot)$ (binary data)	Threshold	Static
Ours	Yes	No	$PC(\cdot,\cdot)$, $ES(\cdot,\cdot)$,$MP(\cdot)$, $TC(\cdot)$	Threshold, Top-κ, progressive	Static, streaming

Table 2 Properties of the supported multivariate correlation metrics

Measure/abbrev	Normalization	Clustering distance
Pearson (*PC*)	Z-norm	Angular
Euclidean (*ES*)	None	Euclidean
Multipoles (*MP*)	Z-norm	Angular
Total (*TC*)	None	Normalized inform. [27]

Our algorithm—Correlation Detective, abbreviated as *CD*—exploits the insight that vectors often exhibit (weak) correlations between each other. For example, securities of companies that participate in the same conglomeration (e.g., Fig. 2a, GOOGL and GOOG) or are exposed to similar risks and opportunities (e.g., STMicroelectronics and ASML) typically exhibit a high correlation between their stock prices. CD exploits such correlations, even if they are weak, to drastically reduce the search space.

CD works as follows: rather than iterating over all possible vector combinations that correspond to the correlation pattern, CD clusters the vectors based on their similarity, and enumerates the combinations of only the cluster centroids. For each of these combinations, CD computes upper and lower bounds on the correlations of all vector combinations in the Cartesian product of the clusters. Based on these bounds, CD decides whether or not the combination of clusters (i.e., all combinations of vectors derived from these clusters) should be added to the result set, can safely be discarded, or, finally, if the clusters should be split into smaller subclusters for deriving tighter bounds. This approach effectively reduces the number of combinations that need to be considered, making CD at least an order of magnitude faster than existing methods.

In the remainder of this section, we will present the key elements of CD, explaining how the two types of queries presented in Sect. 2 are handled. We will start with a brief description of the initialization phase, which includes data pre-processing and clustering. In Sects. 3.2 and 3.3, we will describe how CD answers threshold and top-κ queries, respectively.

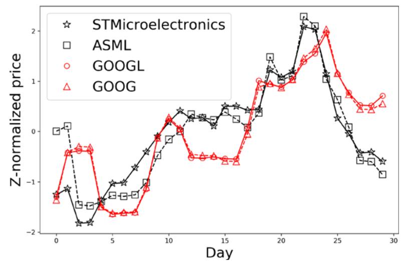
3.1 Initialization and clustering

First, all vectors are normalized using a measure-specific (e.g., *PC*, *ES*, *MP*, *TC*) normalization technique (discussed in Sect. 3.2).

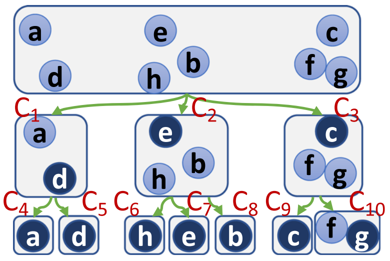
The second part of the initialization phase considers constructing a hierarchical clustering of all vectors, again using a measure-specific distance measure (shown in Table 2). We will discuss the selection of distance measures in Sect. 3.2.2.

The clustering algorithm operates in top-down fashion. A root cluster containing all vectors is first created to initialize the hierarchy. The algorithm then consists of three steps. First, K vectors are picked from the root cluster and used as the initial top-level centroids in the hierarchy. These vectors are picked using the seeding strategy of K-means^{++} [5]. The use of K-means^{++} (as opposed to sampling K random vectors) ensures that these initial centroids are well-distributed over the metric space, and not very close to each other. In the second step, we run the standard K-means algorithm for at most r_1 iterations, or until convergence using the average function to recompute the cluster centroids after each iteration. The clustering is evaluated using the Within-Cluster Sum of Squares (WCSS) (the sum of the variances within all clusters). In the third step, steps one and two are repeated r_2 times (i.e., with different centroids), and the clustering with the lowest WCSS is kept as the final clustering assignment for the first level of the hierarchy. These three steps are executed recursively on each individual cluster with non-zero radius, to construct the second, third, etc. levels of the hierarchy, until all leaf nodes contain only one vector.

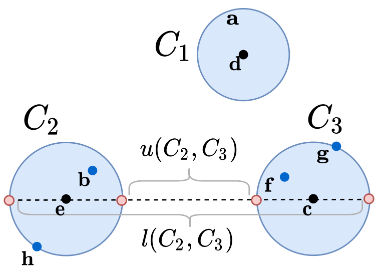
There is a clear tradeoff between the cost of the clustering algorithm and the clustering quality. Increasing the values of r_1 and r_2 will generally result in a higher clustering quality (lower WCSS), but will take longer to compute. However, the quality of the clustering does *not* affect the correctness of CD—in fact, regardless of the employed hierarchical clustering algorithm, CD always returns the same correct result set. A poor clustering only affects the computational efficiency of CD. Still, our experiments show that as long as the clustering is reasonable, a suboptimal clustering is not detrimental to CD's efficiency. More precisely, we found that the value of r_1 (max. iterations of K-means, after the initial centroids



(a)



(b)



(c)

Fig. 2 **a** Two groups of closely related stocks: ASML and STMicroelectronics are exposed to similar risks, while GOOG and GOOGL participate in the same conglomeration; **b** Running example in 2 dimensions: the centroids of each cluster are depicted with darker background. All clusters are labeled for easy reference; **c** Illustration of pessimistic pairwise bounds of Lemma 1

were decided) had no observable effect on CD's efficiency. Therefore, we simply set $r_1 = 1$. The same generally holds for r_2, although to prevent ruinous effects due to coincidentally very poorly chosen initial centroids, we set $r_2 = 50$. Still, the clustering takes at most a few seconds in our experiments, which is negligible compared to the total execution time of the algorithm.

3.2 Threshold queries

CD receives as input the cluster tree produced by the hierarchical clustering algorithm, a correlation pattern, and a correlation threshold τ. It then forms all possible combinations of the correlation pattern with the child clusters of the root. In the example of Fig. 2b, for a desired correlation pattern of $PC(2, 1)$, the following *combinations of clusters* are examined:

$$\forall_{C_x, C_y, C_z \in \{C_1, C_2, C_3\}} ((C_x, C_y), C_z)$$

Note that we now present the algorithm for finding all interesting triplets following correlation pattern $PC(2, 1)$. In reality, CD also considers all sub-patterns of the queried correlation pattern (e.g., $PC(1, 1)$) by re-running the same algorithm on those sub-patterns.

A combination of clusters compactly represents the combinations created by the Cartesian product of the vectors inside the clusters. For example, assuming that $|C_x| = 4$ and $|C_y| = 3$, the cluster combination (C_x, C_y) represents a set of 12 vector combinations, which we will refer to as its *materializations*. For each cluster combination, the algorithm computes lower and upper bounds on the correlation of its materializations, denoted with LB and UB, respectively (Algorithm 1, line 1). These bounds guarantee that any possible materialization of the cluster combination, i.e., replacing each cluster with any one of the vectors in that cluster, will always have a correlation between LB and UB.

The next step is to compare the bounds with the user-chosen threshold τ (lines 2, 4, 6). If $UB < \tau$, the combination

Algorithm 1: THRESHOLDQUERY($\mathcal{S}_l, \mathcal{S}_r, Corr, \tau$)

```
Input: Sets of clusters $\mathcal{S}_l$ and $\mathcal{S}_r$ that adhere to the user-defined
       correlation pattern including a correlation measure $Corr$,
       correlation threshold $\tau$.
1  $(LB, UB) \leftarrow$ CALCBOUNDS($\mathcal{S}_l, \mathcal{S}_r, Corr$)
2  if $LB \geq \tau$ then
3      Add $(\mathcal{S}_l, \mathcal{S}_r)$ to the result set
4  else if $UB < \tau$ then
5      Discard $(\mathcal{S}_l, \mathcal{S}_r)$
6  else
           // Replace largest cluster with subclusters and recurse
7      $C_{max} \leftarrow \arg\max_{C \in \mathcal{S}_l \cup \mathcal{S}_r} \{C.radius\}$
8      Set $SC \leftarrow C_{max}.subclusters$
9      for $S \in SC$ do
10         $(\mathcal{S}_l', \mathcal{S}_r') \leftarrow (\mathcal{S}_l, \mathcal{S}_r)$ with $C_{max}$ replaced by $S$
11         THRESHOLDQUERY($(\mathcal{S}_l', \mathcal{S}_r')$, $Corr, \tau$)
```

is *decisive negative*—no materialization yields a correlation higher than the threshold τ. Therefore, this cluster combination does not need to be examined further. If $LB \geq \tau$, the combination is *decisive positive*, guaranteeing that all possible materializations of this cluster combination will have a correlation of at least τ. Therefore, all materializations are inserted in the result. Finally, when $LB < \tau$ and $UB \geq \tau$, the combination is *indecisive*. In this case, the algorithm (lines 7–11) chooses the cluster $C_{\max}$ with the largest radius,[4] and recursively checks all combinations where $C_{\max}$ is replaced by one of its sub-clusters. In the example of Fig. 2b, assume that the algorithm examined an indecisive combination of clusters C_1, C_2, C_3, and C_2 is the cluster with the largest radius. The algorithm will drill down to consider the three children of C_2, and examine their combinations with C_1 and C_3. The recursion continues until each combination is decisive.

We will refer to this process as *traversing the comparison tree*. Decisive combinations are typically found at high levels of the cluster tree, thereby saving many comparisons. In the

[4] Radii are computed using the distance metrics in Table 2.

following, we will discuss two different approaches for deriving LB and UB for arbitrary correlation patterns. The first approach (theoretical bounds) has constant complexity in the number of materializations a cluster combination covers. The second approach (empirical bounds) extends the theoretical bounds with additional information. It has a slightly higher cost, but typically leads to much tighter bounds.

3.2.1 Theoretical bounds

We first present a lemma for bounding the cosine similarity between only two clusters, which serves as a stepping stone for bounding multivariate correlations.

Lemma 1 *Let* $\cos(\theta_{\mathbf{x},\mathbf{y}})$ *denote the cosine similarity between two vectors* $\mathbf{x}$ *and* $\mathbf{y}$*, with* $\theta_{\mathbf{x},\mathbf{y}}$ *being the angle formed by these vectors. Consider four vectors* $\mathbf{u_1}$*,* $\mathbf{u_2}$*,* $\mathbf{v_1}$*, and* $\mathbf{v_2}$*, such that* $\theta_{\mathbf{v_1},\mathbf{u_1}} \leq \theta_1$ *and* $\theta_{\mathbf{v_2},\mathbf{u_2}} \leq \theta_2$*. Then, cosine similarity* $\cos(\theta_{\mathbf{u_1},\mathbf{u_2}})$ *can be bounded as follows:*

$$\cos(\theta^{max}_{\mathbf{u_1},\mathbf{u_2}}) \leq \cos(\theta_{\mathbf{u_1},\mathbf{u_2}}) \leq \cos(\theta^{min}_{\mathbf{u_1},\mathbf{u_2}})$$

where

$$\theta^{min}_{\mathbf{u_1},\mathbf{u_2}} \max\left(0, \theta_{\mathbf{v_1},\mathbf{v_2}} - \theta_1 - \theta_2\right)$$
$$\theta^{max}_{\mathbf{u_1},\mathbf{u_2}} \min\left(\pi, \theta_{\mathbf{v_1},\mathbf{v_2}} + \theta_1 + \theta_2\right)$$

Proof All proofs are included in Appendix A of the Technical Report [12] □

Lemma 1 bounds the cosine similarity between two vectors $\mathbf{u_1}$ and $\mathbf{u_2}$ that belong to two clusters with centroids $\mathbf{v_1}$ and $\mathbf{v_2}$, respectively, by using: (a) the angle between the two centroids, and, (b) upper bounds on the angles between $\mathbf{u_1}$ and $\mathbf{v_1}$, and between $\mathbf{u_2}$ and $\mathbf{v_2}$. For instance, in the running example (Fig. 2b), we can bound the cosine between **a** and **b** if we have the cosine of the two cluster centroids **d** and **e**, the cosines of **a** with **d**, and of **h** with **e** (as **h** is the furthest point in C_2 from the centroid **e**). The bounds are tightened if the maximum angle formed by each centroid with its corresponding cluster vectors is reduced. We now extend our discussion to cover multivariate correlations, which involve three or more clusters.

Theorem 1 (Bounds for PC) *For any pair of clusters* C_i, C_j*, let* $l(C_i, C_j)$ *and* $u(C_i, C_j)$ *denote lower/upper bounds on the pairwise correlations* ρ *between the cluster pair's materializations, i.e.,* $l(C_i, C_j) \leq \min_{\mathbf{x}\in C_i, \mathbf{y}\in C_j} \rho(\mathbf{x}, \mathbf{y})$ *and* $u(C_i, C_j) \geq \max_{\mathbf{x}\in C_i, \mathbf{y}\in C_j} \rho(\mathbf{x}, \mathbf{y})$*. Consider sets of clusters* $\mathcal{S}_l = \{C^l_i\}^{p_l}_{i=1}$ *and* $\mathcal{S}_r = \{C^r_j\}^{p_r}_{j=1}$*. Let* $L(\mathcal{S}_1, \mathcal{S}_2) = \sum_{C_i\in\mathcal{S}_1, C_j\in\mathcal{S}_2} l(C_i, C_j)$*, and* $U(\mathcal{S}_1, \mathcal{S}_2) = \sum_{C_i\in\mathcal{S}_1, C_j\in\mathcal{S}_2} u(C_i, C_j)$*.*

Then, for any two sets of z-normalized vectors[5] $X = \{\hat{\boldsymbol{x}}_1, \ldots, \hat{\boldsymbol{x}}_{p_l}\}$*,* $Y = \{\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_{p_r}\}$ *such that* $\hat{\boldsymbol{x}}_i \in C^l_i, \hat{\boldsymbol{y}}_i \in C^r_i$*, multivariate correlation* $PC(X, Y)$*, can be bounded as follows:*

1. *if* $L(\mathcal{S}_l, \mathcal{S}_r) \geq 0 : PC(X, Y) \in$
$$\left[\frac{L(\mathcal{S}_l, \mathcal{S}_r)}{\sqrt{U(\mathcal{S}_l, \mathcal{S}_l)}\sqrt{U(\mathcal{S}_r, \mathcal{S}_r)}}, \frac{U(\mathcal{S}_l, \mathcal{S}_r)}{\sqrt{L(\mathcal{S}_l, \mathcal{S}_l)}\sqrt{L(\mathcal{S}_r, \mathcal{S}_r)}}\right]$$
2. *if* $U(\mathcal{S}_l, \mathcal{S}_r) \leq 0 : PC(X, Y) \in$
$$\left[\frac{L(\mathcal{S}_l, \mathcal{S}_r)}{\sqrt{L(\mathcal{S}_l, \mathcal{S}_l)}\sqrt{L(\mathcal{S}_r, \mathcal{S}_r)}}, \frac{U(\mathcal{S}_l, \mathcal{S}_r)}{\sqrt{U(\mathcal{S}_l, \mathcal{S}_l)}\sqrt{U(\mathcal{S}_r, \mathcal{S}_r)}}\right]$$
3. *else:* $PC(X, Y) \in$
$$\left[\frac{L(\mathcal{S}_l, \mathcal{S}_r)}{\sqrt{L(\mathcal{S}_l, \mathcal{S}_l)}\sqrt{L(\mathcal{S}_r, \mathcal{S}_r)}}, \frac{U(\mathcal{S}_l, \mathcal{S}_r)}{\sqrt{L(\mathcal{S}_l, \mathcal{S}_l)}\sqrt{L(\mathcal{S}_r, \mathcal{S}_r)}}\right]$$

As Pearson correlation is equivalent to cosine similarity when computed over z-normalized vectors, we can use Lemma 1 to compute bounds on the pairwise correlations between any pair of clusters, which allows us to compute the bounds in Theorem 1. Consequently, we can bound the multivariate correlation of any cluster combination that satisfies the PC correlation pattern, without testing all its possible materializations. For example, for combination $((C_1, C_2), C_3)$ from our running example, we first use Lemma 1 to calculate bounds for all cluster pairs in $O(1)$ per pair, which leads to values for $L(\cdot, \cdot)$ and $U(\cdot, \cdot)$. The bounds on $PC((C_1, C_2), C_3)$ then follow directly from Theorem 1.

Theorem 2 (Bounds for MP) *For any pair of clusters* C_i, C_j*, let* $l(C_i, C_j)$ *and* $u(C_i, C_j)$ *denote lower / upper bounds on the pairwise correlations between the cluster's materializations, i.e.,* $l(C_i, C_j) \leq \min_{\mathbf{x}\in C_i, \mathbf{y}\in C_j} \rho(\mathbf{x}, \mathbf{y})$ *and* $u(C_i, C_j) \geq \max_{\mathbf{x}\in C_i, \mathbf{y}\in C_j} \rho(\mathbf{x}, \mathbf{y})$*. Consider the set of clusters* $\mathcal{S} = \{C_i\}^p_{i=1}$*. Furthermore, let* $\mathbf{L}$ *and* $\mathbf{U}$ *be symmetric matrices such that* $\mathbf{L}_{ij} = l(C_i, C_j)$ *and* $\mathbf{U}_{ij} = u(C_i, C_j)$ $\forall 1 \leq i, j \leq p$*. For any set of* ***z-normalized*** *vectors* $X = \{\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{x}}_2, \ldots, \hat{\boldsymbol{x}}_p\}$ *such that* $\hat{\boldsymbol{x}}_i \in C_i$*, multipole correlation* $\mathrm{MP}(X)$ *can be bounded as follows:*

$$MP(X) \in 1 - \lambda_{min}\left(\frac{\mathbf{L}+\mathbf{U}}{2}\right) \pm \frac{1}{2}||\mathbf{U}-\mathbf{L}||_2$$

where $\lambda_{min}\left(\frac{\mathbf{L}+\mathbf{U}}{2}\right)$ *is the smallest eigenvalue of matrix* $\left(\frac{\mathbf{L}+\mathbf{U}}{2}\right)$*.*

[5] Z-normalization involves shifting and scaling a vector such that they have zero mean and unit standard deviation.

Similar to Theorem 1 for *PC*, we can use Lemma 1 to compute the bounds on the pairwise correlations between any pair of clusters, which allows us to compute the bounds of Theorem 2, and to analyze the *MP* values of all materializations of the cluster combination in one go.

Theorem 3 (Bounds for ES) *For any pair of clusters C_i, C_j, let $l(C_i, C_j)$ and $u(C_i, C_j)$ denote lower / upper bounds on the dot products $\langle \cdot, \cdot \rangle$ between the clusters' materializations, i.e., $l(C_i, C_j) \le \min_{\mathbf{x} \in C_i, \mathbf{y} \in C_j} \langle \mathbf{x}, \mathbf{y} \rangle$ and $u(C_i, C_j) \ge \max_{\mathbf{x} \in C_i, \mathbf{y} \in C_j} \langle \mathbf{x}, \mathbf{y} \rangle$. Consider the sets of clusters $\mathcal{S}_l = \{C_i^l\}_{i=1}^{p_l}$ and $\mathcal{S}_r = \{C_j^r\}_{j=1}^{p_r}$. Let $L(\mathcal{S}_1, \mathcal{S}_2) = \sum_{C_i \in \mathcal{S}_1, C_j \in \mathcal{S}_2} l(C_i, C_j)$, and $U(\mathcal{S}_1, \mathcal{S}_2) = \sum_{C_i \in \mathcal{S}_1, C_j \in \mathcal{S}_2} u(C_i, C_j)$. Then, for any two sets of vectors $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_{p_l}\}$, $Y = \{\mathbf{y}_1, \ldots, \mathbf{y}_{p_r}\}$ such that $\mathbf{x}_i \in C_i^l$, $\mathbf{y}_i \in C_i^r$, multivariate correlation $ES(X, Y)$, can be bounded as follows: $ES(X, Y) \in$*

$$\left[\left(1 + \sqrt{\frac{U(\mathcal{S}_l, \mathcal{S}_l)}{p_l^2} + \frac{U(\mathcal{S}_r, \mathcal{S}_r)}{p_r^2} - 2\frac{L(\mathcal{S}_l, \mathcal{S}_r)}{p_l p_r}}\right)^{-1}, \left(1 + \sqrt{\frac{L(\mathcal{S}_l, \mathcal{S}_l)}{p_l^2} + \frac{L(\mathcal{S}_r, \mathcal{S}_r)}{p_r^2} - 2\frac{U(\mathcal{S}_l, \mathcal{S}_r)}{p_l p_r}}\right)^{-1}\right]$$

Since $\langle \mathbf{x}, \mathbf{y} \rangle = \cos(\theta_{\mathbf{x},\mathbf{y}}) \|\mathbf{x}\|_2 \|\mathbf{y}\|_2$, we can again use Lemma 1 to compute bounds on $L(\cdot, \cdot)$ and $U(\cdot, \cdot)$, which allow us to compute the bounds of Theorem 3. This is done by first computing bounds on cosines with Lemma 1 for all cluster pairs in $O(1)$ per pair, and combining those with bounds on the l_2-norms of each cluster. [6]

Theorem 4 (Bounds for TC) *For any pair of clusters C_i, C_j, let $l(C_i, C_j)$ and $u(C_i, C_j)$ denote lower / upper bounds on the joint (Shannon) entropy $H(\cdot, \cdot)$ between the clusters' materializations, i.e., $l(C_i, C_j) \le \min_{\mathbf{x} \in C_i, \mathbf{y} \in C_j} H(\mathbf{x}, \mathbf{y})$ and $u(C_i, C_j) \ge \max_{\mathbf{x} \in C_i, \mathbf{y} \in C_j} H(\mathbf{x}, \mathbf{y})$. Similarly, let $l(C_i)$ and $u(C_i)$ denote lower/upper bounds on the marginal entropies of vectors in the cluster C_i. Consider the set of clusters $\mathcal{S} = \{C_i\}_{i=1}^{p}$ with $\mathcal{S}_i$ denoting the i-th cluster in the set. Then, for any set of vectors $X = \{\hat{x}_1, \ldots, \hat{x}_p\}$ such that $\mathbf{x}_i \in C_i$, multivariate correlation $TC(X)$, can be bounded as follows: $TC(X) \in$*

$$\left[\sum_{i=1}^{p} l(C_i) - \sum_{i=1}^{p-1} (\min_{1 \le j \le i} u(C_{i+1}|C_j)) - u(C_1), \sum_{C_i \in \mathcal{S}} u(C_i) - \max_{C_i, C_j \in \mathcal{S}} l(C_i, C_j)\right]$$

[6] Similar to z-normalization for *PC* and *MP*, the l_2-norm of each vector can be computed and cached as a preprocessing step, after which bounds on the norms per cluster can be quickly derived on cluster initialization.

Theorems 1–3 are built on the observation that the multivariate correlation of a set of vectors can be expressed as a function of the pairwise relations exhibited by the vectors in that set. Then, this (exact) expression of a multivariate correlation among *individual* vectors is extended to bounds on the multivariate correlation among *clusters* of vectors, which are in turn bounded by Lemma 1.

Although the Total Correlation of a set of vectors X cannot be expressed as a function of cosine similarities, it can be bounded by other pairwise relations, namely conditional entropies with two variables [35]. This enables us to express bounds the *TC*-value of a set of vectors as a function of correlation bounds between pairs of clusters, similar to the previous Theorems [52]. How these bounds on cluster pairs are computed (and tightened) in the absence of Lemma 1 will be discussed in the following section.

Note that Theorem 4 bounds apply to both discrete and continuous data, using differential entropy for the latter case. In case exact probability functions are unknown for continuous data, one can derive empirical distribution functions through discretization.

3.2.2 Tightening the bounds

Empirical pairwise bounds. The bounds of Lemma 1—which are used for deriving the bounds of Theorems 1, 2, and 3—tend to be pessimistic, as they always account for the worst theoretical case. In the example of Fig. 2c, the theoretical lower bound (resp. upper bound) accounts for the case that hypothetical vectors (depicted in pink) are located on the clusters' edges, resulting in the smallest (resp. largest) possible distance between any pair of points in the clusters.

Tightening the bounds on cosine similarities will in turn tighten the bounds on *PC*, *MP*, and *ES*, which will lead to more aggressive pruning power of the algorithm described earlier in this section. The *empirical bounds* approach builds on the observation that the cosine similarities of any pair of vectors $\mathbf{x_i}, \mathbf{x_j}$ drawn from a pair of clusters C_i, C_j, respectively, is typically strongly concentrated around $(l(C_i, C_j) + u(C_i, C_j))/2$, especially for high-dimensional vectors. The approach works as follows. At initialization, we compute all (pairwise) cosines and store these in an upper-triangular matrix. Then, during execution of Algorithm 1, we compute $l(C_i, C_j)$ and $u(C_i, C_j)$, when required, as follows:

$$l(C_i, C_j) = \min_{\mathbf{x} \in C_i, \mathbf{y} \in C_j} \cos(\theta_{\mathbf{x},\mathbf{y}})$$

and

$$u(C_i, C_j) = \max_{\mathbf{x} \in C_i, \mathbf{y} \in C_j} \cos(\theta_{\mathbf{x},\mathbf{y}})$$

with $\cos(\theta_{\mathbf{x},\mathbf{y}})$ retrieved from the upper-triangular matrix. The computed $l(C_i, C_j)$ and $u(C_i, C_j)$ are also cached and reused whenever (C_i, C_j) is encountered in another cluster combination.

It is important to note that the empirical bounds do not induce errors, since they trivially satisfy the requirements of Theorems 1–3 that $l(C_i, C_j) \leq \min_{\mathbf{x}\in C_i, \mathbf{y}\in C_j} \cos(\theta_{\mathbf{x},\mathbf{y}})$ and $u(C_i, C_j) \geq \max_{\mathbf{x}\in C_i, \mathbf{y}\in C_j} \cos(\theta_{\mathbf{x},\mathbf{y}})$. Therefore, the bounds of multivariate correlations derived using these empirical bounds are still correct. Finally, they are *at least as tight* as the bounds of Lemma 1, since they account only the vectors that are actually present in the clusters and not the hypothetical worst case.

There is a clear tradeoff between the cost of computing the empirical pairwise bounds (worst case, quadratic to the number of vectors), and the performance improvement of CD from the tighter bounds. Indicatively, in our experiments, the theoretical pairwise bounds computed from Lemma 1 were typically between two to eight times wider compared to the empirical pairwise bounds. Exploiting the tighter empirical bounds led to a reduction of the width of the bounds of Theorem 1 by 50% to 90% (for $PC(1, 2)$), which empowered CD to reach to decisive combinations faster. As a result, total execution time of the algorithm with empirical bounds was typically an order of magnitude less than the time with the theoretical bounds. Therefore, all reported results will be using the empirical bounds.

Lastly, note that the empirically-bounded versions of Theorem 1 and 2 do not require z-normalization. Still, it is performed in both cases to optimize pairwise cache computation and to ensure that $MP \in [-1, 1]$, as suggested in [3]. However, z-normalization does not impact relative distances and therefore the top-κ query answers are identical.

Total Correlation bounds. The empirical bounding approach can also be used to compute bounds on the (conditional) entropies between pairs of clusters, which are key in computing the TC bounds of Theorem 4. As $H(A|B) = H(A, B) - H(B)$, this can be done by (a) pre-computing and caching all marginal entropies and (pairwise) joint entropies of vectors, and, (b) iterating over the Cartesian products of clusters to derive bounds on the entropies of cluster materializations.

Notice that the lower bound of $TC(X)$ (see Theorem 4) involves iterating over $\mathcal{S}$ *in sequence*, which indicates a dependency on the ordering of clusters in $\mathcal{S}$. Thereby, finding the optimal permutation of $\mathcal{S}$ that produces the tightest bound will increase the lower bound without introducing errors in the result set. The total number of permutations is $O(p!)$, where p is the number of vectors in the correlation pattern. Here we introduce a heuristic that costs $O(p^2)$. The heuristic, shown in Algorithm 2, computes a tight upper bound on the joint entropy $H(X)$,[7] by iterating over the sorted list of marginal and conditional entropies to find a selection of entropies that closely estimates $H(X)$. Note that, for conciseness, Algorithm 2 line 3 indicates we always fetch a conditional entropy $H(C_i|C_j)$ from the head of the queue $\mathcal{H}$. However, as $\mathcal{H}$ also contains marginal entropies $H(C_i)$, the condition may also be empty.

Algorithm 2: TCPermHeuristic($\mathcal{H}$)

```
Input: A priority queue $\mathcal{H}$ with all marginal and conditional
       entropy upper bounds for a set of clusters $\mathcal{S} = \{C_i\}_{i=1}^p$
Output: Upper bound on the joint entropy of materializations of
        $\mathcal{S}$
1 $U \leftarrow \{\}, H_X = 0$
2 while $|U| < p$ do
3     $H(C_i|C_j) \leftarrow \mathcal{H}$.pop()
4     if $C_i \notin U \wedge C_j \notin U$ then
5         $H_X = H_X + H(C_i|C_j)$
6         $U \leftarrow U \cup C_i$
7 return $H_X$
```

Choosing a distance measure for clustering. The empirical pairwise bounds tighten the bounds on correlations between cluster pairs, leading to also tighter multivariate correlation bounds, and improved efficiency of CD. Tightness of the empirical bounds depends on the cluster radius—clusters with large radii lead to weaker, albeit correct, bounds. This is clear for PC, ES, and MP, where triangle inequality is also present in the theoretical bounds (see Sect. 3.2.1). However, our experiments have shown that tuning the clustering distance measure also benefits TC queries, even though TC does not satisfy the triangle inequality. Therefore, the clustering distance measure always impacts the pruning power of the algorithm.

As Lemma 1 is based on angular distance, clustering for PC and MP employs the clustering loss function (WCSS) with angular radii. For ES, Euclidean distance is the obvious choice, since it also considers vector norms, which are not captured with the angular radii but are included in Theorem 3. Finally, for TC, our experiments showed that the normalized information distance metric $D(X, Y) = 1 - \frac{I(X,Y)}{H(X,Y)}$ (first introduced in [27]) leads to tight multivariate correlation bounds. The intuition behind this observation is that $D(X, Y)$ measures information proximity, similar to TC—in fact, $D(X, Y)$ is simply a transformation of the pairwise total correlation (i.e., mutual information) between two variables to a strict distance metric ranging between 0 and 1 [26]. Table 2 summarizes these choices.

[7] $H(X)$ is upper bounded by the factor $\sum_{i=1}^{p-1}(\min_{1\leq j\leq i} u(C_{i+1}|C_j)) - u(C_1)$ in $TC_{LB}(X)$.

3.2.3 Handling of additional constraints

CD supports both the irreducibility and minimum jump constraints, as described in Sect. 2. For irreducibility, the process of identifying whether a simpler combination exists requires testing whether a combination of any of the subsets of $\mathcal{S}_l$ and $\mathcal{S}_r$ is already contained in the answers.

To avoid the cost of enumerating all $O(2^{|\mathcal{S}_l|+|\mathcal{S}_r|})$ subsets during the execution of Algorithm 1, only the pairwise correlations between any two clusters $C_l \in \mathcal{S}_l$ and $C_r \in \mathcal{S}_r$ are examined.

Precisely, we use $l(C_l, C_r)$, which is already computed for Theorems 1–4. If there exist C_l, C_r s.t. $l(C_l, C_r) \geq \tau$, then any solution that can be derived from further examining the combination $(\mathcal{S}_l, \mathcal{S}_r)$ cannot satisfy the irreducibility constraint. Therefore, $(\mathcal{S}_l, \mathcal{S}_r)$ can be discarded. The case of minimum jump is analogous: if any $l(C_l, C_r) \geq UB - \delta$, where UB is calculated as in line 1 of Algorithm 1, then the combination is discarded.

By considering only the pairwise correlations during the pruning process may lead to inclusion of answers that do not satisfy the constraints. Such combinations are filtered from the query result before returning it to the user. Since the number of answers is typically in the order of a few tens to thousands, this final pass takes negligible time.

Both *MP* and *TC* have the property that correlation can only increase when adding an extra variable (i.e., $TC(X \cup \{y\}) \geq TC(X)$. We refer to this property as the *monotonicity over increasing pattern length*. This reduces the relevance of *MP* and *TC* threshold queries without any constraints, as for any $TC(X) \geq \tau$ with $X \subset \mathcal{V}$, all supersets of X will be in the result set, making it more cluttered. Therefore, we disallow such queries for *MP* and *TC*, defaulting to the addition of the irreducibility constraint. Note that we could still answer unconstrained queries on *MP* and *TC*, essentially cost-free, by expanding the result set $\mathcal{R}$ as follows:

$$\{X \cup A : A \subseteq \mathcal{V}, X \in \mathcal{R} \big| |A| \in [1, p - |X|] \subset \mathbb{N}^+\}$$

However, we refrain from doing so as these additional results do not provide new insights to the user.

3.3 Top-k queries

The top-κ variant addresses this issue by allowing users to set the desired number of results, instead of τ. The answer then includes the κ combinations of vectors with the highest correlation that satisfy the correlation pattern.

Assuming an oracle that can predict the τ that would yield exactly κ results, the top-κ queries could be transformed to threshold queries and answered with the standard CD algorithm. Since such an oracle is impossible, many top-κ algorithms (e.g., Fagin's threshold algorithm [16]) start with a low estimate for τ, and progressively increase it, by observing the intermediate answers. The performance of these algorithms depends on how fast they can approach the true value of τ, thereby filtering candidate solutions more effectively.

The top-κ variant of CD (see Algorithm 3) follows the same idea. The algorithm has the same core as the threshold-based variant, and relies on *three* techniques to rapidly increase τ.

Top-κ pairwise correlations First, at initialization, input parameter τ is set to the value of the κ'th highest pairwise correlation. Since all pairwise correlations are computed for the empirical bounds, this causes zero additional cost.

Exploiting (soft) monotonicity. The second technique is inspired by the property of monotonicity of *MP* and *TC*, which implies that multivariate correlations can only increase when adding an additional variable (i.e., vector) to the set (i.e., correlation pattern). Thereby, given the top-κ combinations of size s, $\mathcal{R}_s$, one can guarantee that any combination of size $s+1$ that is a superset of a combination in $\mathcal{R}_s$ will have a correlation greater than the lowest correlation in $\mathcal{R}_s$, and will lead to an increase of threshold τ.

This observation is exploited by exhaustively computing the correlations of all possible supersets of size $s+1$ after finding $\mathcal{R}_s$, in order to quickly increase τ before traversing the comparison tree with combinations of size $s+1$ to construct $\mathcal{R}_{s+1}$. This technique showed to be very effective for all correlation measures (despite *PC* and *ES* not possessing the monotonicity property), as many of the supersets of $\mathcal{R}_s$ were also included $\mathcal{R}_{s+1}$.

Prioritization of candidates The last technique is an optimistic refinement of the upper bound, aiming to prioritize the combinations with the highest correlations. The algorithm is executed in two phases. In the first phase, similar to Algorithm 1, the algorithm traverses the comparison tree in a Breadth-First manner (BFS), and computes the upper and lower bound per combination. However, it now artificially tightens the bounds by decreasing the value of the upper bound as follows;

$$UB_{\text{shrunk}} = (1-\gamma)\frac{UB+LB}{2} + \gamma UB$$

where $\gamma \in [-1, 1]$ is a shrink factor parameter with a default value of 0. Now, decisiveness of cluster combinations is determined based on (LB, UB_{shrunk}) analogous to Algorithm 1, with an exception of the case where $UB_{\text{shrunk}} \leq \tau < UB$ (Algorithm 3 lines 3,7,12). In this case, the cluster combination is *postponed* for further inspection, and placed in a priority queue based on the combination's *critical shrink factor* γ^*—the minimum value of γ for which UB_{shrunk} surpasses τ (lines 12–14). Intuitively, a small γ^* means that the combination (i.e., branch in the comparison tree) is more

Algorithm 3: TOP-κ-QUERY($\mathcal{S}_l, \mathcal{S}_r, Corr, \tau, \kappa, \gamma$)

```
Input: Sets of clusters $\mathcal{S}_l$ and $\mathcal{S}_r$ that adhere to the user-defined
       correlation pattern. correlation measure $Corr$, starting
       threshold $\tau$, desired output set size $\kappa$, shrinkfactor $\gamma$.
1  $(LB, UB_{shrunk}) \leftarrow$ CALCBOUNDS($\mathcal{S}_l, \mathcal{S}_r, Corr, \gamma$)
2  $B \leftarrow$ new priority queue
3  if $LB \geq \tau$ then
4      Add the contents of $(\mathcal{S}_l, \mathcal{S}_r)$ to the result set $\mathcal{R}$
5      $\mathcal{R} \leftarrow$ SORT($\mathcal{R}$)[1:$\kappa$]
6      $\tau \leftarrow \min_{(X,Y)\in\mathcal{R}} Corr(X, Y)$
7  else if $UB_{shrunk} \geq \tau$ then
       // Replace largest cluster with subclusters and recurse with
       TOP-$\kappa$-QUERY (similar to lines 7–11 of Algorithm 1)
12 else
13     $\gamma^* = \frac{\tau-\mu}{UB-\mu}$
14     $B$.ADD (($\mathcal{S}_l, \mathcal{S}_r$), $\gamma^*$)
   // Phase 2 - starts when Phase 1 is
   completed for $(\mathcal{S}_l, \mathcal{S}_r) \in B$ do          // Traverse
   comp-tree DFS
15     THRESHOLDQUERY($\mathcal{S}_l, \mathcal{S}_r, Corr, \tau$)
16 $\mathcal{R} \leftarrow$ SORT($\mathcal{R}$)[1:$\kappa$]
17 $\tau \leftarrow \min_{(X,Y)\in\mathcal{R}} Corr(X, Y)$
```

promising to lead to higher correlation values as a large portion of its bound range ($UB - LB$) exceeds τ. In the second phase (lines 15–18), postponed branches are traversed in a Depth-First manner (DFS) by invoking Algorithm 1 on each combination sequentially. Since τ continuously increases, and the first branches are likely to contain the highest correlation values, most lower-priority branches do not need many cluster splits to reach decisive combinations. Similar to the previous optimizations, the value of γ only impacts efficiency of the algorithm, and not completeness of the results. Our experiments (see Sect. 5) have shown that values of gamma around 0 lead to a good balance between DFS and BFS exploration.

3.4 Progressive queries

The prioritization technique of Algorithm 3 can also be used as a basis for a progressive threshold algorithm. Precisely, Algorithm 3 can be initialized with a user-chosen τ and with $\kappa \to \infty$. This will prioritize the combinations that will yield the strongest correlations, and thus also the majority of correlations larger than τ. Prioritization is frequently useful in exploratory data analytics: the user may choose to let the algorithm run until completion, which will yield results identical to Algorithm 1, or interrupt the algorithm after receiving sufficient answers. Recent work also established accurate (any-time) prediction of result completeness and distance for kNN queries [14]. Although valuable, their methods require significant adaptations for our queries and are thus deferred to future work. We evaluate CD on all proposed query types in Sect. 5.2.

4 Detection of multivariate correlations in streaming data

Data is frequently observed as a live stream. For example, in finance, asset prices may need to be monitored in real-time for detecting strong correlations in a market, for portfolio diversification [39]. In weather monitoring, real-time detection of correlations may reveal interesting short-term weather events, whereas in server monitoring, detection of unexpected correlations, e.g., on server requests originating from many different IP addresses, may reveal attempts of attacks [44]. Similarly, in neuroscience, real-time analysis of fMRI streams to detect correlations brings novel exploitation opportunities, e.g., for neurofeedback training [22, 31, 54].

Our streaming algorithm, called CDStream, builds on top of CD such that it maintains CD's solution over a sliding window as new data arrives. CDStream does this efficiently by storing the decisive cluster combinations in a custom index, which can subsequently be used after each streaming update to quickly identify the potential changes to the result set. Clearly, the main challenge is to construct, maintain, and utilize this index efficiently, for processing streams with high update rates. CDStream supports *PC* and *ES* correlation measures. In the remainder of this section, we will explain the underlying stream processing model and CDStream algorithm in detail. We will also present an extension to CDStream named CDHybrid, which dynamically switches between CDStream and repeated execution of CD in order to adapt to sudden events and concept drift, and improve robustness.

4.1 Stream processing model

CDStream builds on the basic windows model, which is widely used for processing of data streams, e.g., in [19, 24, 51, 53]. The model works as follows: the sliding window, of length w, is partitioned to a set of smaller, fixed-length sub-windows (often called *basic windows*), each of length b. All stream updates received within a basic window are processed (typically aggregated), to generate a single value for that basic window. In other words, the basic windows define the time resolution handled by the algorithm.

The introduction of basic windows offers several benefits: (a) it makes the results robust to outliers, noise in the data, and time series with small-period oscillations, e.g., stocks with high trading volumes, (b) it allows for handling time-misaligned and out-of-order arrivals, which are fairly common in real-life data streams (e.g., stock ticks, sensors with variable measurement intervals, weak/slow network connections), and (c) it allows efficient handling of streams with high update rates. At the same time, this approach introduces a—potentially significant—delay on the results, which can be as large as b time units. The latter constraint becomes

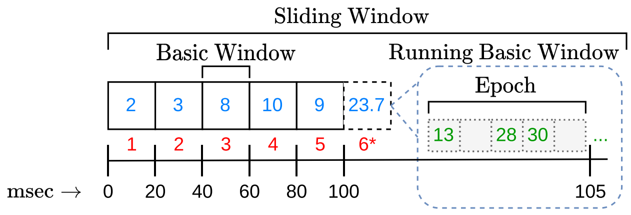


Fig. 3 Example of a stream representation with the BW$^+$ model with $w = 100$, $b = 20$, $epoch = 5$. With red we denote the index/position of the basic window. The blue numbers correspond to the values of the corresponding windows. The updates in the running basic window and running epoch are shown in green color

Algorithm 4: HandleEpoch($S, A, C, \mathcal{I}, \tau$)

```
Input: Set of streams $S$, set of arrivals $A$ , pairwise correlations
       cache $C$, DCC Index $\mathcal{I}$, correlation threshold $\tau$
1 for $(i, v) \in A$ do          // (i: stream id, v: value)
2     Recompute $S_i$'s last basic window's aggregate
3     for $j = 1$ to $n$ do      // Update pairwise cache C
4         $C[i, j] \leftarrow Corr(S_i, S_j)$
5 for $(i, v) \in A$ do          // Check for violations
6     $V \leftarrow$ QueryIndex($i, \mathcal{I}$)      // Query violations
7     for $(\mathcal{S}_l, \mathcal{S}_r) \in V$ do    // Recompute and re-index
8         ThresholdQuery($\mathcal{S}_l, \mathcal{S}_r, Corr, \tau$)
9         UpdateIndex($\mathcal{S}_l, \mathcal{S}_r, Corr, \tau, \mathcal{I}$)
```

limiting when processing periods of high activity (e.g., in high-volatility periods of a stock market, or when a network is under a DDoS attack), where it is critical that the user observes intermediary results as soon as possible.

CDStream alleviates this limitation by disentangling the period of recomputing the results (the key reason behind the stale results) with the length of the basic window b. The model, called BW$^+$ hereafter, offers an extra knob to the user, called *epoch*, which controls the acceptable delay/lag for the algorithm to account for new data. When *epoch* is set to be equal to b, BW$^+$ degenerates to the standard basic windows model, e.g., as used in [53]. However, by setting *epoch* to be less than b, the algorithm is instructed to recompute the results more than once within the period of a basic window, accounting also for the new arrivals in the incomplete basic window. The aggregation unit remains unchanged, i.e., the basic window of size b, which allows meaningful handling of time misalignment, noise and outliers. Furthermore, all completed basic windows are not impacted by the epoch—hence their aggregate values are not recomputed. However, whenever an epoch is completed, the algorithm updates the aggregate value for the incomplete basic window and updates the multivariate correlations, to include these new values.

As an example, consider the stream depicted in Fig. 3. Assume that *epoch* is set to 5 msec, and the basic and sliding window lengths, b and w, are set to 20 and 100 msec, respectively. Then, at time 100, BW$^+$ will have identical results to the standard basic windows model. At time 105, BW$^+$ will recompute the results, accounting for the values that arrived in basic windows 1–5, and within the first five seconds of the (still incomplete) basic window 6. Therefore, if in the period between time 100 and 105, there were drastic changes that led to updates of the results, these will be detected by BW$^+$. The same process will be repeated at times 110 and 115, whereas at time 120, basic window 1 will expire and the results of BW$^+$ will again become identical to the output of the standard basic windows model (not shown in figure). It is important to note that BW$^+$ with an epoch less than b is not equivalent to running the standard basic windows algorithm with $b = epoch$. BW$^+$ keeps the completed basic windows intact—it does not change their boundaries when an epoch is complete. As we will explain in the following section, this is leveraged by CDStream to optimize performance by avoiding to store, or recompute, fine-grained partial results. We will come back to the discussion about the properties of BW$^+$, and its impact in terms of computational efficiency and accuracy/completeness of the results of the algorithm in Sect. 4.4.

Time-based vs arrival-based epoch. Even though our previous discussion assumed that epochs are defined in time units (seconds, minutes, etc.), this does not constitute a requirement of the model. Epochs can also be defined in number of arrivals (e.g., every 10 arrivals). A definition based on number of arrivals may be preferred in use cases where the arrival rate of the streams changes abruptly, e.g., during a market crash.

4.2 Algorithm core

We start with a high-level description of CDStream before going over the details of the underlying custom index, which is instrumental for increasing the throughput of the algorithm. CDStream receives as an input the set of streams, and the configuration parameters of the algorithm—length of the sliding window w and basic window b, *epoch*, and query threshold. The algorithm starts by executing CD on the last w arrivals in the given streams, and prints the initial results to the user. A byproduct of CD is an upper-diagonal matrix that stores the pairwise correlations between all pairs of streams. We will refer to this as the pairwise correlations cache. Then, CDStream enters the monitoring phase. In this phase, whenever an epoch is completed, the algorithm (shown in Algorithm 4) first detects all streams that have at least one update and recomputes the corresponding aggregate for the last (potentially still incomplete) basic window (line 2). It then refreshes the cache of pairwise correlations, to account for the new arrivals (lines 3–4). Notice that this step does not recompute the correlations from scratch; it updates them from the previous correlation values and the change in aggregate value for the running basic window. Following, the algorithm goes through all updates within the epoch, and checks whether these could lead to changes in the result set (either new additions in the result or removals). This process is supported by a custom-build index, which returns all decisive cluster combinations with bounds impacted by the newly arrived updates. These impacted bounds are then reassessed using Algorithm 1, in order to detect the potential changes in the result set, and to update the index (Algorithm 4 lines 5–9).[8] The described steps are repeated for $\lfloor epoch/b \rfloor$ epochs, after which a basic window is completed. In that case, CDStream will additionally remove the expired basic window, add the newly-completed basic window, and keep repeating the above process (not depicted in Algorithm 4).

In the remainder of this section we will look at the custom index, and how this is maintained and utilized by CDStream.

The DCC index In short, the index is used for storing a collection of thresholds, that, when fired, signify a potential change in the answer set.[9] Particularly, the core idea is to store decisive cluster combinations (abbreviated as DCCs) for all clusters, and enable re-validating only these after every stream update. Recall that each stream **s** belongs to a hierarchy of clusters. For example, vector **e** in Fig. 2b belongs to C_2 and C_7. For a stream **s**, we denote the set of these clusters as $\mathcal{C}(\mathbf{s})$. By construction, the algorithm takes a decision concerning any stream **s** based solely on the decisive combinations including any cluster in $\mathcal{C}(\mathbf{s})$ (see the theoretical results in Sect. 3.2.1). As long as those decisive combinations are still valid, the final result will remain correct and complete.

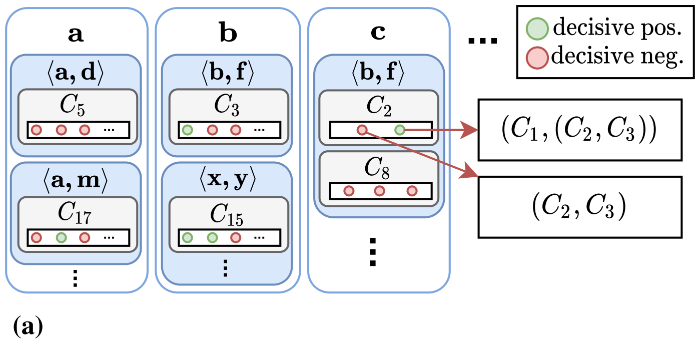
A naive approach would be to construct an inverted index that maps each cluster to the decisive cluster combinations it participates in. Then, after any update of a stream **s**, we would look at all clusters in $\mathcal{C}(\mathbf{s})$, and find and re-validate all their decisive combinations from the index. The use of this index could become too slow for some use cases, particularly for large correlation patterns, due to the potentially large number of decisive combinations associated with each cluster that need to be checked. Two key observations can be exploited to optimize the use of this index: (a) the empirical correlation bounds described in Sect. 3.2.2 do not depend on all streams contained in the cluster, but are determined solely by $l(C_i, C_j)$ and $u(C_i, C_j)$, the minimum and maximum *pairwise* correlations between all involved clusters in the combination, and (b) the previous applies independent of the number of the clusters contained in the left and right side of the cluster combination. Therefore, the DCC index is designed around these minimum and maximum pairwise correlations.

Figure 4a depicts an example of the internal organization of the DCC index. At the outer layer, the index is an inverted index that maps each stream **s** to a list of extrema pairs. A pair of streams is called an extremum pair if there exists at least one cluster combination for which this pair constitutes a determining pair, i.e., it is the pair determining the value of $l(C_i, C_j)$ or $u(C_i, C_j)$. For example, in Fig. 2c, the minimum and maximum extrema pairs for (C_2, C_3) are $\langle \mathbf{h}, \mathbf{g} \rangle$ and $\langle \mathbf{b}, \mathbf{f} \rangle$, determining the minimum value $l(C_i, C_j)$ and maximum value $u(C_i, C_j)$, respectively. At the inner layer, for each extremum pair *ep* we keep a list of all *opposite clusters*, i.e., the clusters that do not include **s**, and participate in at least one decisive cluster combination having *ep* as an extremum pair. For example, focusing at **c** in Fig. 4a, we see that one of its extrema pairs is $\langle b, f \rangle$, which is reused by both clusters C_2 and C_8. The clusters are stored in decreasing size, i.e., the cluster at position $i+1$ will be a sub-cluster of the cluster at position i. For each cluster, we store all decisive combinations, and whether these are positive or negative. In our running example, for cluster C_2 we have a negative combination (C_2, C_3) and a positive combination $(C_1, (C_2, C_3))$. This way of indexing and querying ensures that we only re-validate DCCs with an actual change in bounds, and that this set is complete (i.e., we do not miss any violations).

When an update is observed at stream **s**, the first step is to use the index for retrieving all extrema pairs that involve a cluster in $C(\mathbf{s})$. For each extremum pair, we check the pairwise cache whether the pair has changed as a result of the last update. This will happen, e.g., if the update of **s** has caused **s** to form a new extremum pair with another stream, replacing an older pair. If the extremum pair has not changed, we can skip all contents grouped under this pair altogether. In our running example, if **c** has been updated, but $\langle \mathbf{b}, \mathbf{f} \rangle$ is still a valid extremum pair for cluster C_2, no further validations are needed for any of the combinations involving C_2. Furthermore, no validations are required for the combinations involving C_8 (and any other clusters following C_2 with the same extremum pair), since C_8 is a strict subset of C_2 (recall that the clusters are ordered based on their size). If, on the other hand, the update has invalidated an extremum pair, the algorithm drills into the contents of the inner layer, and goes over the clusters sharing this extremum pair. If, e.g., **c** was updated and $\langle \mathbf{b}, \mathbf{f} \rangle$ is no longer an extremum pair for C_2, we need to check and adjust all combinations stored for C_2 (in this example, (C_2, C_3) and $(C_1, (C_2, C_3))$). This is done by adjusting the extrema pairs and bounds using Theorems 1 and 3, re-validating whether the combination is still decisive—positive or negative, and updating the solution accordingly. In this step, the algorithm may even need to break a cluster to two or more sub-clusters, until it again reaches to decisive combinations. However, again, as soon as we find a cluster for which the extremum pair does not change after the update, we can move to the next extremum pair.

[8] In practice, method UPDATEINDEX is coded inside a custom implementation of Algorithm 1, to avoid duplicate work.

[9] Similar indices were used in earlier works, e.g., [33], but for bounding the values of individual correlations.



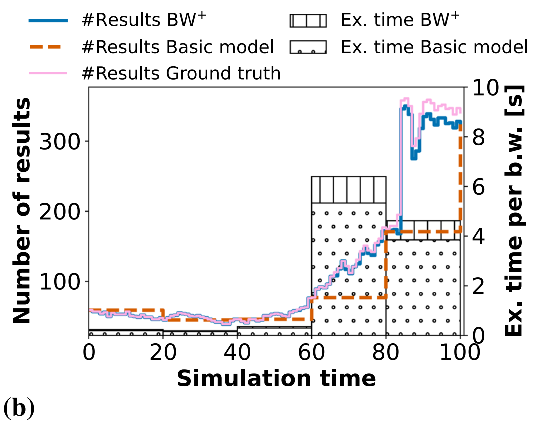


Fig. 4 **a** Visualization of the decisive combination index; **b** Number of results and execution time per basic window, with BW$^+$ and the standard basic window model. BW$^+$ is configured with epoch size 1. The results correspond to the Stocks dataset, with $n = 1000$, $w = 120000$, and $b = 20$

4.3 User constraints and top-κ queries

To support the minimum jump and irreducibility constraints, additional triggering functionalities, further described below, are added to the index of CDStream.

Irreducibility constraint. Let X, Y, X', Y' denote sets of clusters. Consider combinations (X, Y), and $(X' \subseteq X, Y' \subseteq Y)$, with $|X \cup Y| > |X' \cup Y'|$, i.e., irreducibility excludes (X, Y) from the results if (X', Y') is in. We need to detect two additional cases: (1) (X, Y) needs to be removed from the result set because (X', Y') just surpassed τ, and, (2) (X, Y) needs to be added in the result set, because (X', Y') was just removed from the result set because its correlation dropped below τ. Both cases can be triggered by an update of a vector from X or Y (hence, also from X' and Y').

Without the irreducibility constraint, the index contains the following extrema pairs: (a) for the negative decisive combinations, the pairs required for upper bounding the correlation, (b) for the positive decisive combinations, all pairs required for lower-bounding the correlation. The irreducibility constraint requires also monitoring of the upper bounds of positive decisive combinations (e.g., for case (1), when an increase of $Corr(X', Y')$ will cause the following condition to hold: $Corr(X', Y') > \tau$ which will mean that (X, Y) need to be removed from the result set) and the lower bounds of negative decisive combinations with any $Corr(X', Y') > \tau$. These decisive combinations are also added in the index, under the extrema pairs, and checked accordingly.

Minimum jump constraint. Monitoring for the minimum jump constraint is analogous to the irreducibility constraint. The following cases need to be considered: (1) (X, Y) needs to be removed from the result set because $Corr(X', Y') + \delta > Corr(X, Y)$, and (2) (X, Y) needs to be added in the result set because $Corr(X', Y') + \delta < Corr(X, Y)$. Both cases are identified using the discussed method for monitoring the irreducibility constraint.

Top-κ queries Recall that CDStream is initialized with the result of CD. For a top-κ query, CDStream queries CD for a slightly larger number of results $\kappa' = b_k * \kappa$, where b_k is a small integer, greater than 1. CDStream finds the minimum correlation in these results, and uses it as a threshold τ in the streaming algorithm. As long as the size of the result set is at least κ, the true top-κ results will always have a correlation higher than τ and will be contained in the top-κ' results maintained by the algorithm. Therefore, the top-κ out of the detected top-κ' correlations are returned to the user.

Scaling factor b_k controls the tradeoff between the robustness of the streaming algorithm for top-κ queries, and its efficiency. Setting $b_k = 1$ may lead to the situation that, due to an update, fewer than κ results exist with correlation greater than or equal to τ. CDStream then fails to retrieve enough results, and resorts to CD for computing the correct answer, and updating its index. Conversely, a large b_k will lead to a larger number of intermediary results, and to more effort for computing the exact correlations of these results, which is necessary for retaining the top-κ results. Our experiments with a variety of datasets have shown that $b_k = 2$ is already sufficient to provide good performance without compromising the robustness of CDStream. We evaluate CDStream in Sect. 5.3.

4.4 Impact of the extended basic window model on CDStream

Recall that CDStream leverages the proposed extended basic window stream processing model (abbrev. as BW$^+$) in order to identify updates on the result set earlier. By construction, BW$^+$ is *at least as good* as the standard basic windows model in terms of completeness of the result set, since it replicates its behavior every time a basic window is completed. The further improvement that we can expect from BW$^+$—compared to the standard basic windows model—depends on the volatility of the input streams. In periods where the input streams contain negligible changes, BW^+ will detect very few additional correlations (if any), compared to the standard model. In periods of high volatility, such as market crashes, BW^+ will detect updates and new correlations faster.

To examine the importance of BW^+ and evaluate its impact on the computational efficiency of CDStream, we compared the results of CDStream, with and without BW^+. Figure 4b presents the number of results (left axis) and runtime (right axis) of CDStream of the two models. The results correspond to processing of a stream with minute-granularity stock prices of 1000 stocks on 16 March 2020 (the dataset is further described in Sect. 5). This day was selected because it was the day of the largest price drop in the 2020 Covid crash [1]. As ground truth, we used the results of CD on the same input dataset (without basic windows), recomputed at the end of each epoch.

We see that BW^+ is able to identify jumps in the number of results significantly earlier than BW. Comparison with the ground truth revealed that BW^+ maintained a recall of 97.8% during this period while BW recall decreased to 69.0%. From epoch 0 to 60 (prior the crash), the recall of BW^+ was 100%.

It is also interesting to consider execution time per basic window. Since the new model subsumes the basic window model, it is slightly more expensive to maintain. However, extra computation is only around 10%, for the more-detailed epoch. This extra computation can of course be adjusted, by increasing the epoch length. Therefore, all experiments hereafter will only focus on the BW^+ model.

4.5 CDHybrid: combining CD and CDStream

Recall that CDStream handles the stream updates in epochs. The algorithm exhibits high performance when the updates do not drastically change the results set. In streams where the answer changes abruptly, it may be more efficient to simply run CD after the completion of each epoch and recompute the solution from scratch, instead of maintaining CDStream's index and the result through time. CDHybrid is an algorithm that orchestrates CD and CDStream, transparently managing the switch between the two algorithms based on the properties of the input stream.

To decide between CD and CDStream, CDHybrid needs to estimate the cost of both approaches for handling an epoch. A good predictor for this is the number of arrivals in the epoch—more arrivals tend to cause more changes in the result, which takes longer for CDStream to handle. Therefore, CDHybrid starts with a brief training period, where it collects statistics on the observed arrival count and execution time of the two algorithms. Simple (online) linear regression is then used to model the relationship between execution time and the observed number of arrivals. Note that the coefficients of a simple linear regression model can be maintained in constant time and space. Therefore, the regression model is continuously updated, even after the training phase. Switching from one algorithm to the other works as follows.

Switching from CDStream to CD. We cache the current results of CDStream (we will refer to these as $\mathcal{R}_{\text{CDStream}}$) and stop maintaining the index. When an epoch is completed, the vectors are updated and passed to CD for computing the result.

Switching from CD to CDStream. Since the stream index was not updated for some time, we need to update it before we can use it again. We compute the symmetric difference Δ of the current results of CD (denoted as $\mathcal{R}_{\text{CD}}$) with the last results of CDStream $\mathcal{R}_{\text{CDStream}}$. Any result r contained in $\Delta \cap \mathcal{R}_{\text{CDStream}}$ is due to a positive decisive combination that has now become negative, whereas any r contained in $\Delta \cap \mathcal{R}_{\text{CD}}$ leads to a new positive decisive combination. In both cases, the algorithm updates the index accordingly. There is also the case that a decisive combination becomes indecisive. In this case, the algorithm recursively breaks the combination further, as shown in Algorithm 1. We evaluate CDHybrid in Sect. 5.3.3.

5 Evaluation

The purpose of our experiments was twofold: (a) to assess the scalability and efficiency of our methods, and, (b) to compare them to a series of baselines. The baselines include the state-of-the-art algorithms for multivariate correlation discovery [2, 3], two variants of an exhaustive search algorithm, as well as multiple modern database management systems (DBMS) that can be used to detect multivariate correlations. Our evaluation does not consider the practical significance of multivariate correlations, as this was already extensively demonstrated in earlier works and case studies from different domains, e.g., [2, 3, 29] (see Sect. 1 for more examples). Still, to ensure that our evaluation is conducted on data where detection of multivariate correlations has practical significance, we also compare our methods with the data used in these past case studies (or data of the same type, where the original data was inaccessible).

Hardware and implementations. All experiments, except for the comparison with the DBMS systems, were executed on a server equipped with two Xeon E5-2697v2 12-Core 2.70 GHz processors, and 500GB RAM. For CoMEtExtended and CONTRa, we used the original implementations, which were kindly provided by the authors [2, 3]. We additionally configured the implementation of CoMEtExtended in order to use all available cores of our server. Consequently, all implementations, except CONTRa and two DBMS, were multi-threaded. Algorithm performance comparisons are exclusively made under matching execution styles (e.g., comparing single-threaded CD only to CONTRa and DBMS). All implementations, except of the UNOPT exhaustive search

Table 3 Default parameters for the experiments with static and streaming data

Dataset	Static					Streaming						
	n	d	κ	γ	K	n	Epoch size	Basic window size	w(h)	Aggr. method	Threshold *PC*	Threshold *ES*
Stocks	1440	1309	100	0	10	1000	1 min	2 h	2000	Sum	0.95	0.15
fMRI	1440	5470	100	0	10	1440	1 s	1 s	0.5	Last	0.9	0.12
SLP	1440	2927	100	0	10	1000	1 h	6 h	2160	Avg	0.99	0.7
TMP	1440	2927	100	0	10	1000	1 h	6 h	2160	Avg	0.99	0.7
Crypto	1440	713	100	0	10	1000	1 min	1 h	216	Sum	0.95	0.15
Deep	1440	96	100	0	10	–	–	–	–	–	–	–

baseline, cached and reused the pairwise correlation computations, using our results presented in Sect. 3.2. This caching was always beneficial for performance. The reported execution time for CD and CDStream corresponds to the total execution cost including the steps of normalizing, clustering and calculating pairwise correlations. All reported results correspond to medians after 10 repetitions. Due to permission constraints on the server, the DBMS experiments were executed on another machine, with an Intel i7-10750H 12-Core 2.60 GHz processor, 32GB RAM, running Ubuntu 22.04.1 LTS.

Datasets. We present extensive evaluation results on seven datasets, coming from distinct disciplines (neuroscience, finance, crypto trading, climate science, and machine learning). See GitHub for download links, pre-processing steps, instructions, and code for reading and processing the data.

- *Stocks.* Daily closing prices of 28678 stocks over the period Jan. 2, 2016 to Dec. 31, 2020 leading to 1309 observations. For the streaming experiments, we used the minute closing prices of the stocks.
- *fMRI.* Functional MRI data of a person watching a movie.[10] Five datasets were extracted by mean-pooling the data with kernels of different size, leading to 237, 509, 1440, 3152, and 9700 time series, respectively, all of 5470 observations. A similar dataset was used in the case study of [2].
- *SLP & TMP.* Segment of the ISD weather dataset [37] containing sea level pressure (**SLP**) and atmospheric temperature (**TMP**) readings of 3222 sensors. CD was evaluated on the daily average values between January 1, 2016 and December 31, 2020, leading to 2927 readings per time series. Streaming experiments were run on hourly sensor measurements.
- *SLP-small.* Sea Level Pressure data [25], as used in the case study of [3]. The dataset contains 171 time series, each with 108 observations.
- *Crypto.* 3-hour closing prices of 7075 crypto-currencies, each with 713 observations, covering the period from April 14, 2021 to July 13, 2021. Streaming experiments were run on minute-level closing prices.
- *Deep.* A billion vectors of length 96, obtained by extracting the embeddings from the final layers of a convolutional neural network [41].

[10] Available online at https://openneuro.org/datasets/ds002837/versions/2.0.0. We used file `sub-1_task-500daysofsummer_bold_blur_censor`, which already includes the recommended pre-processing for voxel-based analytics.

Whenever needed, we obtain subsets of these datasets with random sampling. To avoid repetition, in the following we will mention the experimental configuration only when this deviates from the default configuration, described in Table 3. The remaining section is organized as follows. We start with a comparison of our methods to the baselines (Sect. 5.1), and then conduct an extensive sensitivity analysis of CD (Sect. 5.2) and CDStream (Sect. 5.3).

5.1 Comparison to the baselines

We start by comparing CD to the baselines: (a) two algorithms based on exhaustive search, (b) commercial and open-source modern database management systems, (c) CoMEtExtended [3], and (d) CONTRa [2]. Our experiments compare both efficiency and recall of all systems for threshold queries.

Comparison to exhaustive search baselines No other solution covers CD's range of queries and correlations. For reference on the complexity of the problem, we constructed two baselines (UNOPT and OPT) that exhaustively compute all multivariate correlations by iterating over all possible combinations of vectors. OPT reuses cached pairwise correlations (exploiting our results presented in Sect. 3.2), whereas UNOPT recomputes them for every combination. This comparison only focuses on execution time, as all methods have perfect precision and recall.

Figure 5 plots the time required from CD, UNOPT, and OPT to execute a threshold query on different subsets of

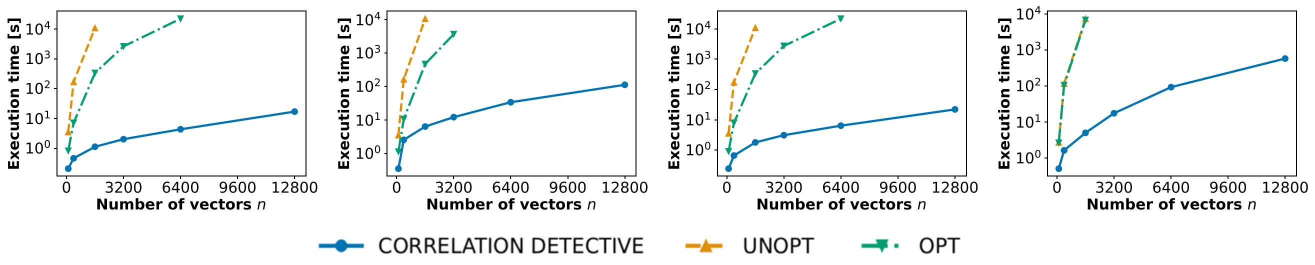


Fig. 5 Scalability of CD and exhaustive baselines for threshold queries on subsets of Stocks. Notice that the Y axis is in logarithmic scale. **a** $ES(1, 2)$, $\tau = 0.85$; **b** $MP(3)$, $\tau = 0.85$; **c** $PC(1, 2)$, $\tau = 0.85$; **d** $TC(3)$, $\tau = 1.7$

Stocks, with sizes up to 12,800 vectors. All algorithms were given at most 8 h to complete. The thresholds were selected such that all correlation measures bring approximately the same number of results on each dataset. Our first observation is that execution time of CD grows at a much slower rate compared to both baselines, for all correlation measures. Furthermore, the difference in efficiency increases with dataset size, which stresses the importance of having an efficient solution like CD. Therefore, CD can handle significantly larger datasets than the baselines.

Comparing OPT to UNOPT, we see that caching of the pairwise correlations improves performance for *ES*, *PC*, and *MP*, but not for *TC*. This is because *TC* is not amenable to the caching optimization, i.e., the *TC* of three or more vectors cannot be expressed as a linear combination of the pairwise *TC* values. Yet, even for the other three measures, OPT still times out for larger datasets. The fact that CD scales better than OPT indicates that its core performance boost comes from the way it utilizes the cluster bounds.

Comparison to contemporary DBMS CD's operation can be expressed as an SQL query, as shown in Fig. 6a, which shows a $PC(1, 2)$ threshold query on a (z-normalized) table named "fmri" in SQL. This observation allows us to compare performance of CD with general-purpose state-of-the-art RDBMS. Our experiment used four off-the-shelf databases, all configured with RAM-stored tables for equitable evaluation, given CD's RAM usage. DBMS1 and DBMS3 supported array attributes, so we developed array functions for Pearson correlation calculation. The other DBMSs stored data in long format (with columns corresponding to a primary key, vector id, time, and value), utilizing a SPSVERBc1 clause for Pearson correlation. Due to limited multi-threading support, all approaches, including CD, ran single-threaded with an eight-hour query limit.

Figure 6b shows the execution times for each system to detect $PC(1, 2)$ on different resolutions of the fMRI dataset. The reported DBMS times do not include the one-off costs of loading the dataset in the DBMS and creating the indices. We see that CD outperforms all DBMS by several orders of magnitude, and the difference between CD and the baselines increases with dataset size. In particular, time complexity for

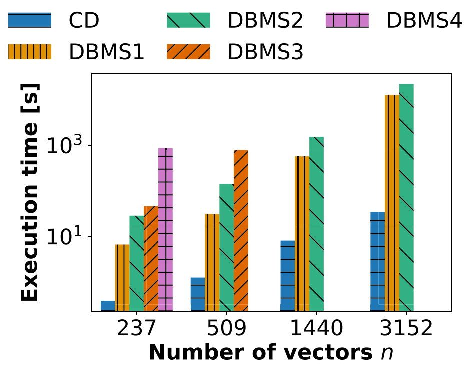


Fig. 6 **a** $PC(1, 2)$ threshold query, implemented with SQL. The correlation measure is implemented as a stored function. **b** Comparison of CD with contemporary DBMS, $PC(1, 2)$, $\tau = 0.8$, $\delta = 0.1$, fMRI

all DBMS seems to follow $O(n^3)$ for performing a triple nested loop (n is the number of vectors), whereas CD's execution time grows at a much slower rate. Furthermore, the results indicate that all DBMS perform similarly to an exhaustive search algorithm, iterating over the full search space.

Comparison to CoMEtExtended Our next experiment was designed to compare CD with CoMEtExtended [3]. CoMEtExtended's goal differs slightly from our problem statement. First, CoMEtExtended is approximate without guarantees. Still, its recall can be tuned by parameter ρ_{CE}, which takes values between -1 and 1. Values around 0 offer a reasonable tradeoff between efficiency and recall [3]. In contrast, CD delivers complete answers, making execution time and recall both relevant in our comparison. Second, CoMEtExtended focuses on *maximal* strongly correlated sets, whereas CD finds *all* such sets (up to a specified cardinality). To ensure a fair comparison for CoMEtExtended, we also considered all subsets of the sets returned by CoMEtExtended. When a subset of a CoMEtExtended answer satisfied the query, we added it to the results, thereby increasing CoMEtExtended's recall. This post-processing step was not included in the execution time of CoMETExtended, i.e., it did not penalize its performance. Table 4 presents the number of results and execution time of CoMEtExtended and CD on the same dataset (SLP-small) and the configuration parameters used in [3].

Table 4 Comparison of CD with CoMEtExtended on SLP-small dataset: execution time (s) and number of retrieved results

τ, δ	CoMEtExtended						Correlation detective			
	$\rho_{CE} = 0$		$\rho_{CE} = 0.01$		$\rho_{CE} = 0.02$		*MP*(4)		*MP*(5)	
	Time	#res	Time	#res	Time	#res	Time	#res	Time	#res
0.4, 0.1	604	62663	1318	67110	3530	70921	7	71083	1132	88305
0.4, 0.15	511	7244	1218	7300	3393	7343	5	7559	579	7562
0.4, 0.2	501	2166	1210	2171	3327	2174	4	2183	248	2183
0.5, 0.1	459	30632	1099	33718	2836	36457	5	34592	635	51391
0.5, 0.15	398	3646	1006	3702	2760	3745	4	3961	355	3964
0.5, 0.2	390	1434	1006	1439	2701	1442	3	1451	193	1451
0.6, 0.1	246	7823	598	8892	1592	9859	3	9204	289	17349
0.6, 0.15	223	1569	577	1606	1559	1635	3	1840	177	1843
0.6, 0.2	219	771	568	776	1532	779	2	788	112	788

Table 5 Comparison of CD with CONTRa on fMRI dataset ($n = 9700$): execution time (s) and number of retrieved results

CONTRa time (#res)	CD ($\tau = 0$) time (#res)	CD ($\tau = 0.5$) time (#res)	CD ($\tau = 0.9$) time (#res)
$\delta = 0.1$:			
$>$ 24 h (23e6)	11510 (23e6)	1908 (21e6)	401 (432)
$\delta = 0.15$:			
11160 (73e4)	4927 (73e4)	1569 (73e4)	391 (102)
$\delta = 0.2$:			
5324 (21e3)	1983 (21e3)	1281 (21e3)	441 (24)

We only consider the *MP* measure, since CoMEtExtended does not support the other three measures.

We see that CD is consistently faster than CoMEtExtended —at least an order of magnitude—and often returns substantially more results. Indicatively, for *MP*(4), CoMEtExtended with $\rho_{CE} = 0$ (resp. $\rho_{CE} = 0.02$) is one to two (resp. two to three) orders of magnitude slower than CD. Notice that for queries with $\delta = 0.1$, $\rho_{CE} = 0.02$ and $\tau = 0.4$, CoMEtExtended also found 281 results with 6 vectors, and one with 7. These amount to $\sim 0.3\%$ of the total amount of discovered results. These were not discovered by CD, which was executed with $p_l = 5$, prioritizing the simpler and more interpretable results. Nevertheless, even for these settings, CD still found 25% more results than COMEtExtended, and in one third of the time. Moreover, the case studies presented in [2, 3] amongst others on this dataset demonstrate the usefulness and significance of relatively simple relationships, involving at most four time series. Other works on multivariate correlations also emphasize the discovery of relationships that do not contain too many time series [9]. For these cases, with a fixed $l_{\max}$, CD is guaranteed to find a superset of COMEtExtended's result set, at a fraction of its cost.

Comparison to CONTRa We also compared CD to CONTRa [2] for discovery of tripoles (i.e., $PC(1, 2) \geq \tau$). To ensure a fair comparison, CD was parameterized to find the same results as CONTRa and to utilize the same hardware, as follows: (a) CD was executed with $\tau = 0$, i.e., pruning was solely due the minimum jump constraint, and (b) CD was configured to utilize at most one thread/core, since the implementation of CONTRa was single-threaded. CONTRa was configured to return exact results.

Table 5 includes execution time and number of results per method.[11] We see that CD is more efficient than CONTRa for detecting identical results, even with $\tau = 0$. However, $\tau = 0$ yields an impractically large amount of results. As such, we also evaluate CD with $\tau = 0.5$ (corresponding to the lowest correlation reported in the case studies of [2]), and with $\tau = 0.9$, which gives a more reasonable amount of results. This further decreases the execution time of CD by one to two orders of magnitude, while preventing an overwhelming number of results.

Summary. Comparison of CD with two state-of-the-art algorithms, two exhaustive baselines, and four DBMS demonstrates that CD consistently outperforms all competitors, requiring at least an order of magnitude less time. This enables CD to find more complex query patterns on larger datasets.

5.2 CD on static data

The following experiments are designed to evaluate the efficiency of CD under different conditions (configurations, datasets, and queries). We first examine the impact of CD's configuration parameters (the shrink factor and the clustering distance) to CD's efficiency. We do not consider recall, since CD is exact, always giving complete answers. Then,

[11] For this experiment, the minimum jump parameter δ is defined as in [2], to represent the minimum difference between the *squared* correlations.

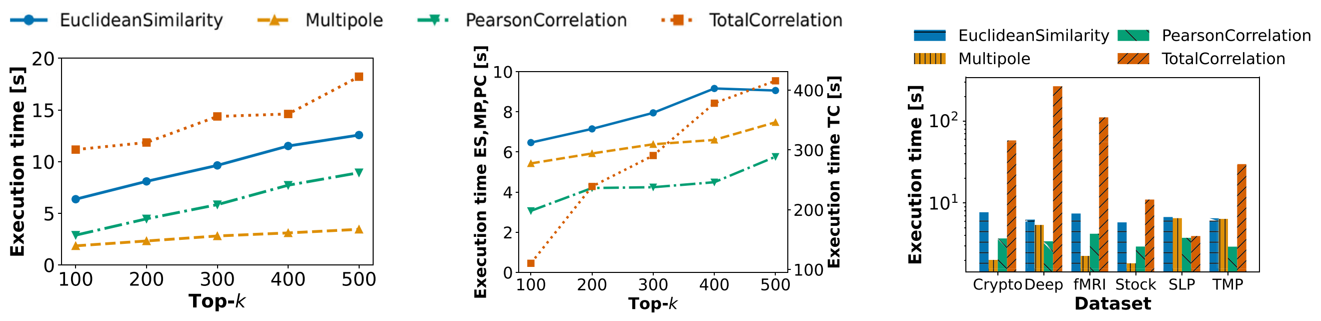


Fig. 7 Effect of κ values and dataset on execution time, with $ES(1,2)$, $MP(3)$, $PC(1,2)$, $TC(3)$. **a** Effect of κ, Stocks; **b** Effect of κ, fMRI; **c** Top-κ on all datasets

we evaluate the performance of CD for top-κ and threshold queries.

5.2.1 Optimizing configuration parameters

We also tested the impact of the values of γ and K (shrink factor and number of sub-clusters per cluster) on CD's efficiency for different configurations. The results showed that both very small ($\gamma = -0.8$) and very large ($\gamma = 0.8$) shrink factor values lead to sub-optimal performance of CD (roughly 38–72% slower than the optimal γ value), as they delay the increase of the running threshold τ. Similarly, extreme values of K also led to sub-optimal performance, with executions being as much as 2x slower for *ES*, *PC*, and *MP* queries, and up to 4x slower for *TC* queries, compared to the execution times with optimal K values. Detailed results can be found in the technical report [12]. However, setting $\gamma = 0$ and $K = 10$ led to near-optimal performance at all configurations—at most 17% worse than the optimal performance in each case. Therefore, for the following experiments we set $\gamma = 0$ and $K = 10$.

5.2.2 Top-κ queries

Effect of κ. Figure 7a, b show the execution time of CD for different values of κ for Stocks and fMRI. We see that a decrease of κ typically leads to increased efficiency. A low value of κ allows for a rapid increase of the running threshold τ, leading to more aggressive pruning at Algorithm 1, line 4. Interestingly, this effect is not equally visible among all considered correlation measures. For example, a reduction of κ gives a significant boost to *ES*, but a much smaller boost for *MP*. This discrepancy is attributed to the correlation values in the result set and the tightness of the bounds. Indicatively, in this experiment, the lowest *MP* value in the result set only decreases from 0.998 (top-100) to 0.9972 (top-500) on the Stocks dataset. In contrast, the lowest *ES* value in the result set decreases from 0.694 (top-100) to 0.672 (top-500) on the same dataset.

Effect of the correlation pattern Table 6 presents execution time of CD for different correlation patterns. As expected, increasing the complexity of the correlation pattern leads to an increase of the computational time. However, even though the size of the search space follows $O\left(\binom{n}{p_l+p_r}\right)$, execution time of CD grows at a much slower rate. Indicatively, for the fMRI dataset, the search space size grows 5 orders of magnitude between $PC(1,2)$ and $PC(1,4)$, whereas CD's execution time increases by only three orders of magnitude, indicating efficient pruning of the search space.

Experiments with different datasets Fig. 7c shows the execution time of CD for all correlation measures on different datasets. We see that efficiency of CD does not vary significantly for *ES* and *PC*. However, performance for queries involving *TC* fluctuates significantly across datasets. This is again attributed to the inherent characteristics of the datasets: analysis of the distributions of the multivariate correlation values in the datasets revealed that the correlations in each dataset followed gamma-like distributions. For *TC*, it is sometimes the case that the mean of this distribution is very close to the minimum correlation in the answer set, i.e., the correlation of the top-κ'th answer. In other words, total correlation is not sufficiently discriminating on these datasets. These situations could be prevented by performing exploratory analysis on the correlation value distribution of a small sample of the dataset. If this analysis does not indicate exceptionally high correlations in the dataset, the data analyst could opt for an alternative correlation measure.

5.2.3 Threshold queries

Effect of threshold Figure 8 shows the effect of threshold τ on the execution time of CD for the Stocks (left Y-axis) and fMRI dataset (right Y-axis) for each correlation measure, and for different constraints. Our first observation is that increasing the threshold leads to higher efficiency for all correlation measures and both datasets. This is expected, since a higher threshold enables more aggressive pruning of candidate comparisons: the upper bounds derived by Theorems 1–4 will be

Table 6 Execution times of CD with different correlation patterns on Top-κ queries (seconds)

Pattern/dataset	ES					MP		PC					TC	
	(1,2)	(1,3)	(1,4)	(2,2)	(2,3)	(3)	(4)	(1,2)	(1,3)	(1,4)	(2,2)	(2,3)	(3)	(4)
fMRI	7	31	4819	168	14,251	6	743	4	19	2808	20	8303	111	11
Stocks	6	12	629	39	2039	6	626	3	7	466	9	1112	12	1570

below τ more often, leading to less recursions. For similar reasons, the addition of stronger constraints (i.e., higher δ or introduction of the irreducibility constraint) generally leads to better performance due to increased pruning power. Furthermore, CD is noticeably faster for *PC* compared to *MP* for the same τ values. This is due to two reasons: (a) the high complexity of the computation of eigenvalues of a matrix (cubic to p_l), which is required for computing the bounds for *MP* (Theorem 2), and, (b) *MP* typically results in higher correlation values and more answers for the same value of τ compared to *PC*.

Progressive variant of CD It is desired for progressive algorithms to collect the majority of results quickly, in order to give early insights to the user about the results, and to enable them to modify/adjust their queries. To evaluate this characteristic of progressive CD (Sect. 3.4), we modified our code such that it saves the discovered results at different time points, and compared these intermediary results with the ground truth, in order to compute recall. In this set of experiments, we focused exclusively on queries which take significant time to complete, since these are the ones that would mostly benefit from a progressive algorithm.

Figure 9 plots the number of results returned by progressive CD at different time points, for all correlation measures on the Stocks dataset. We see that for all correlation measures, CD retrieves more than 90% of the results in the first few seconds – less than 10% of the total execution time. This property of CD is particularly appealing for cases where approximate results will suffice.

Summary. The default configuration parameters for CD (number of clusters and shrink factor) provide near-optimal performance. Complexity of CD grows at a much slower rate compared to size of the search space, and CD is more efficient in scenarios where the chosen correlation measure and threshold are discriminating for the dataset. Finally, progressive CD retrieves more than 90% of the results within the first few seconds.

5.3 CDStream on streaming data

The third set of experiments was designed to evaluate the performance of CDStream. We used the timestamps contained in all datasets (except Deep, which did not contain the notion of time) for generating the streams. Hereafter, we will present detailed results for the Stocks dataset, and include results with the other datasets only when these provide additional insights. We start with experiments with a time-based epoch definition (Sect. 5.3.1), and then investigate the performance of CDStream using arrival-based epochs (Sect. 5.3.2). In the technical report [12] we present additional experiments, including an analysis of the algorithm's performance when executed for a prolonged time period, and an analysis of the impact of the sliding window size on CDStream's efficiency.

5.3.1 Experiments with time-based epochs

Effect of number of streams Figure 10a presents the average processing time per epoch of CDStream, for different numbers of streams. Since there is no streaming baseline for CDStream, the plot includes the average execution time taken by CD, per epoch, to compute the answers, using the same sliding window data (of course, repeated executions of CD are needed to maintain the results with the streaming updates). We see that CDStream is more efficient than CD for small correlation patterns, requiring only a few milliseconds per epoch—an order of magnitude less compared to CD for both correlation measures. Also note that, even though the search space grows at a combinatorial rate with the number of vectors, the growth in execution time of CDStream is substantially slower. This is attributed to the grouping technique in the CDStream index, which effectively reduces the work for processing each update. Also notice that CD outperforms CDStream on more complex correlation patterns. This is because of the index maintenance cost of CDStream: for more complex correlation patterns, the number of combinations that need to be maintained in the index also grows, eventually surpassing the performance boost coming from the index. Since CD does not depend on this index, it avoids this cost. This observation clearly demonstrates the importance of an automated algorithm (similar to the hybrid algorithm proposed in Sect. 4.5) that can dynamically switch between the two for optimizing performance.

Effect of the query parameters Table 7 presents the effect of τ and additional constraint values (minimum jump and irreducibility) on CDStream's performance. We see that efficiency of CDStream is robust to constraints—a constraint only causes a small difference in the number of decisive com-

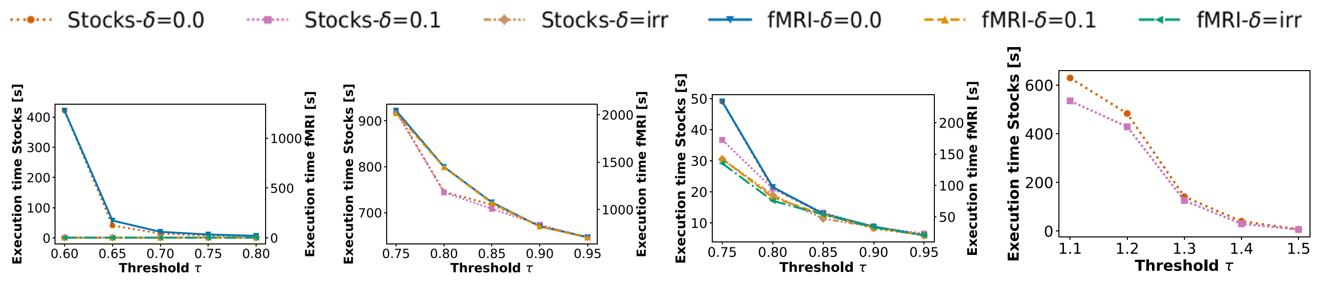


Fig. 8 Effect of constraint and τ on query performance (Stocks and fMRI), **a** $ES(2, 2)$; **b** $MP(4)$; **c** $PC(2, 2)$; **d** $TC(3)$

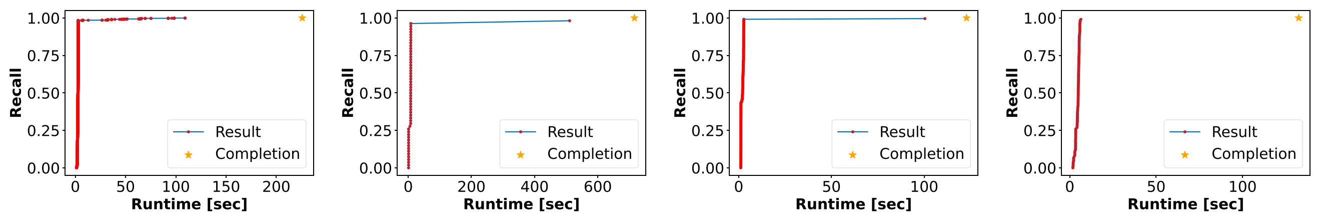


Fig. 9 Number of retrieved results over execution time, for progressive execution of queries on the Stocks dataset. **a** $ES(1, 3)$ query, $\tau = 0.58$, $\delta = 0.03$; **b** $MP(4)$ query, $\tau = 0.8$, $\delta = 0.05$; **c** $PC(1, 3)$ query, $\tau = 0.7$, $\delta = 0.12$; **d** $TC(3)$ query, $\tau = 0.25$, $\delta = 1.3$

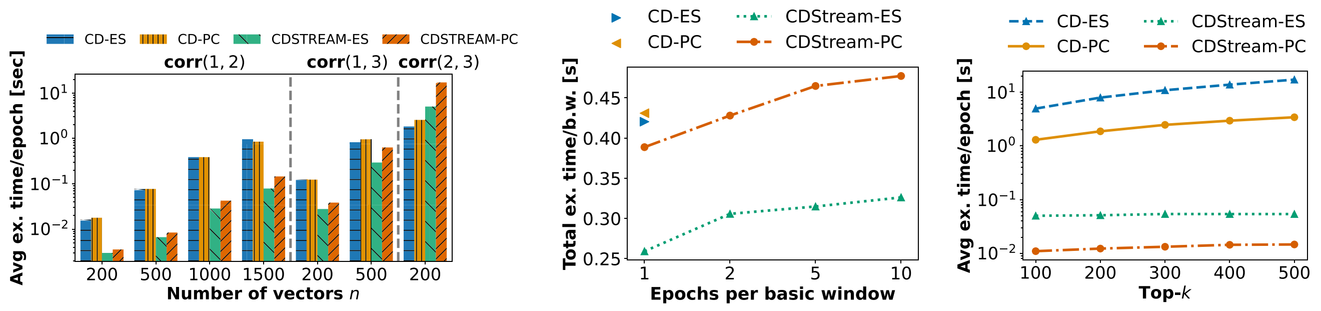


Fig. 10 **a** Effect of dataset size and correlation pattern, with $\tau = 0.95$, Stocks, **b** Effect of epoch size (time-based), $PC(1, 2)$ with $\tau = 0.95$, Stocks, **c** Effect of top-κ, $PC(1, 2)$, Stocks

binations that need to be monitored. In contrast, an increasing value of τ leads to better performance, as decisive combinations are reached earlier, similar to the case of CD.

Effect of epoch size For the next experiment, we fixed the basic window size to 10 min, and measured the processing time per basic window (i.e., sum of epoch execution times), for different epoch sizes. Since the basic window size is fixed, epoch size also determines the number of epochs per basic window. The results, presented in Fig. 10b for the Stocks dataset, demonstrate that CDStream utilizes larger epochs to increase efficiency: larger epochs (alternatively, fewer epochs per basic window) allow CDStream to optimize the checks on the affected DCCs, by combining multiple updates and checking the affected DCC only once. Furthermore, a larger epoch increases the probability that arrivals with outlier values (potentially due to noise)—which would otherwise cause

Table 7 Effect of τ and δ on CD and CDStream's average execution time per epoch (in seconds) with streaming data, Stocks

		CD					CDSTREAM				
	$\tau \setminus \delta$	0.0	0.05	0.1	0.15	irr	0.0	0.05	0.1	0.15	irr
$ES(1, 2)$	0.10	0.416	0.396	0.385	0.378	0.377	0.036	0.036	0.032	0.029	0.025
	0.15	0.402	0.382	0.381	0.372	0.372	0.032	0.030	0.030	0.027	0.027
	0.20	0.402	0.369	0.362	0.360	0.360	0.031	0.029	0.029	0.029	0.029
$PC(1, 2)$	0.80	1.025	0.903	0.772	0.756	0.723	0.234	0.226	0.214	0.206	0.206
	0.90	0.441	0.426	0.412	0.412	0.402	0.085	0.079	0.069	0.069	0.070
	0.95	0.370	0.366	0.365	0.346	0.342	0.051	0.045	0.045	0.046	0.045

temporary invalidations of DCCs—are dampened by other arrivals on the same stream.

We also see that, for all configurations with *ES*, CDStream requires less cumulative time per basic window to maintain the results, compared to a single execution of CD at the end of the basic window. In other words, CDStream updates the results more frequently compared to CD (up to 10 times more frequently in this experiment), and still requires less total execution time. With *PC*, CDStream with 1 and 2 epochs per basic window has comparable performance with a single execution of CD. Increasing the number of epochs further enables CDStream to provide even more frequent updates compared to CD, yet with a slight degradation of efficiency (up to 20% more time). This discrepancy of results for the two correlation measures is due to the inherent distribution of the correlation values—the results for *PC* change more rapidly compared to the results for *ES* in this dataset, which causes a higher cost for maintaining the index.

Top-κ queries Figure 10c plots the average processing time per epoch for top-κ queries $PC(1, 2)$ and $ES(1, 2)$, for different κ values. The results correspond to the Stocks dataset with 1000 stocks. We see that processing time for both algorithms increases with κ for both correlation measures, but at a much slower rate for CDStream compared to CD. In CD, execution time grows almost linearly with κ (from 4.94 to 17.20 s for $ES(1, 2)$), whereas for CDStream the time increases by only 7% for the same queries. The reason for this notable difference in efficiency is that CDStream only maintains the top-κ solutions, already having a good estimate for the threshold of the top-κ highest correlation from previous runs, whereas CD has to start each run from scratch. Therefore, for CDStream, the only increase in execution time for higher κ-values comes from updating and sorting a slightly larger result set and buffer set.

5.3.2 Experiments with arrival-based epochs

Effect of epoch size Figure 11a plots the average processing time per arrival, for varying epoch sizes. As a reference, the plot also includes the average processing time for a periodic re-execution of CD after the end of every epoch (amortized on the epoch's arrival).

We see that increasing the epoch size also increases CDStream's performance. This behavior is expected, since a larger epoch provides more opportunities to CDStream for reducing the number of DCCs that need to be checked. Therefore, similar to the case of time-based epochs (Sect. 5.3.1), epoch size provides a knob to the user for fine-tuning the trade-off between freshness of results and CPU/total execution time.

Also observe that the execution time per arrival for CD approaches that of CDStream as the epoch size increases. In the case of *PC*, the processing time for the two algorithms

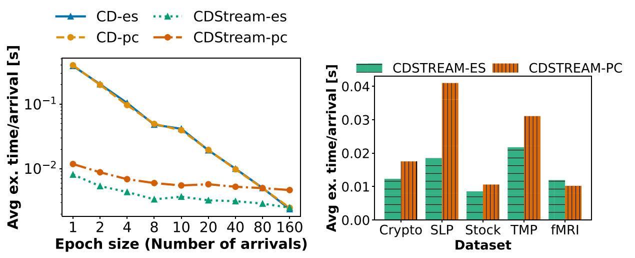


Fig. 11 Effect of query parameters on CDStream's performance with an arrival-based epoch. **a** Effect of epoch size, with $\tau = 0.95$, Stocks; **b** Effect of dataset

crosses at epoch size 80, whereas for *ES*, this crossing happens at epoch size 160. This difference is due to the inherent distribution of correlations according to the two correlation measures in this dataset.

Effect of dataset Figure 11b presents the average execution time per arrival (i.e., epoch size of 1), for $PC(1, 2)$ and $ES(1, 2)$ threshold queries on all datasets. The cost of a periodic execution of CD at the end of every epoch is also included in the figure, as a reference. We see that, even though arrivals are processed in at most 50 msec, processing cost is noticeably higher for the two weather sensor datasets (SLP and TMP) compared to all others. This can be attributed to the lower time resolution in these two datasets(minimum arrival rate for these datasets is 1 h, compared to seconds/minutes for the others). This leads to a substantially higher volatility of the result set, and consequently, to more frequent updates in the DCC index.

5.3.3 Evaluation of CDHybrid

For the final set of experiments, we test the ability of CDHybrid to switch seamlessly and efficiently between CD and CDStream, in order to minimize processing cost in the presence of stream bursts. Since our streams did not present significant bursts that could cause noticeable differences to CDStream throughout the runtime of CDStream, we introduced an artificial burst at all streams between epochs 70 and 90, by temporarily increasing the arrival rate by a factor of 30 (i.e., speeding up all streams during these epochs). CDHybrid was allowed a small warmup period of 40 epochs, where it was processing the updates, but was also switching between CD and CDStream in order to collect initial measurements and train the cost regression model.

Algorithm effectiveness Figure 12a depicts the processing time per epoch (moving averages over 5 epochs), for processing Stocks with CD, CDStream, and CDHybrid. The figure also includes the number of arrivals within each epoch (right Y axis). We observe that when the burst starts—at around epoch 70—CDStream becomes substantially slower than CD, whereas performance of CD is not impacted. CDHybrid immediately recognizes the burst and switches to CD,

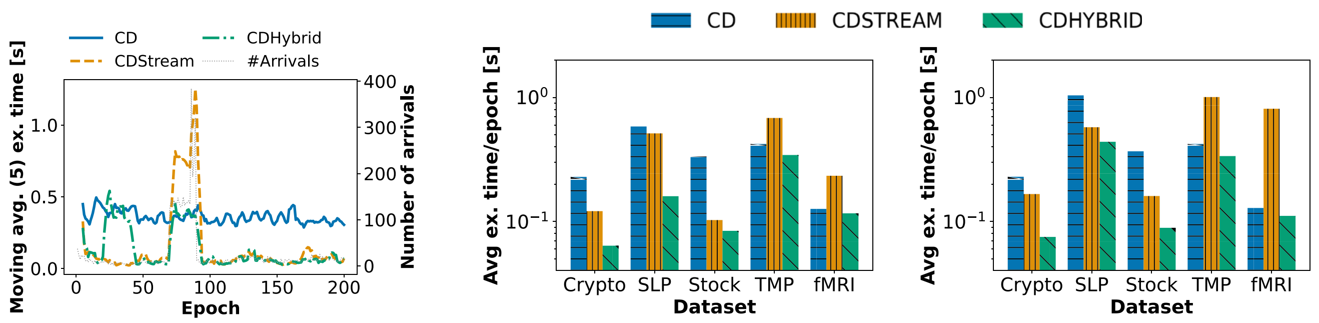


Fig. 12 **a** Efficiency of CDHybrid over time, $PC(1,2)$, $n = 1000$, Stocks, **b** Impact of dataset on CDHybrid efficiency, $ES(1,2)$, **c** Impact of dataset on CDHybrid efficiency, $PC(1,2)$

thereby maintaining peak performance. After the burst is completed (shortly after epoch 90), CDHybrid switches back to CDStream. This switch includes a small additional overhead for updating the DCC index. However, this overhead is insignificant.

Effect of dataset Figure 12b, c show the average processing time per epoch for CD, CDStream, and CDHybrid on all datasets (excluding the warm-up time), for $ES(1,2)$ and $PC(1,2)$ queries, respectively. We see that CDHybrid consistently outperforms both CD and CDStream. This means that neither CD nor CDStream is the best algorithm for processing the whole stream. Yet, CDHybrid efficiently switches between the two as a response to the varying arrival rate, thereby providing near-optimal performance for each epoch.

Summary. CDStream outperforms CD in most scenarios for processing of streams. Epoch size provides a useful knob to the user, for balancing throughput of CDStream with freshness of results. Finally, CDHybrid seamlessly combines the execution of CD and CDStream, offering consistently better performance than both.

6 Conclusions

We considered the problem of detecting high multivariate correlations with four correlation measures, and with different constraints. We proposed three algorithms: (a) CD, optimized for static data, (b) CDStream, which focuses on streaming data, and (c) CDHybrid for streaming data, which autonomously chooses between the two algorithms. The algorithms rely on novel theoretical results, which enable us to bound multivariate correlations between large sets of vectors. A thorough experimental evaluation using real datasets showed that our contribution outperforms the state of the art typically by an order of magnitude.

The current methods are limited to correlations over linear combinations of vectors. Future work should extend them to also accommodate nonlinear aggregations like MIN and MAX, which find applications in the discussed domains. Furthermore, detailed analysis showed that caching pairwise statistics (through ’empirical bounds’) greatly boosted CD’s performance. While all proposed measures suited these bounds, future ones might not. Thus, optimizing the application of the more general theoretical bounds will be vital as the proposed techniques evolve.

Acknowledgements This work has received funding from the European Union’s Horizon Europe research and innovation programme STELAR under Grant Agreement No. 101070122.

References

1. 2020 stock market crash - wikipedia. https://en.wikipedia.org/wiki/2020_stock_market_crash
2. Agrawal, S., Atluri, G., Karpatne, A., Haltom, W., Liess, S., Chatterjee, S., Kumar, V.: Tripoles: a new class of relationships in time series data. In: Proceedings of the SIGKDD’17
3. Agrawal, S., Steinbach, M., Boley, D., Chatterjee, S., Atluri, G., Dang, A.T., Liess, S., Kumar, V.: Mining novel multivariate relationships in time series data using correlation networks. TKDE **32**(9), 1798–1811 (2020)
4. Alemi, A.A., Fischer, I., Dillon, J.V., Murphy, K.: Deep variational information bottleneck. In: ICLR’17
5. Arthur, D., Vassilvitskii, S.: K-Means++: the advantages of careful seeding. In: Proceedings of the SODA’07
6. Carlborg, Ö., Haley, C.S.: Epistasis: Too often neglected in complex trait studies? Nat. Rev. Genet. **5**(8), 618–625 (2004)

7. Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., Abbeel, P.: Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In: NIPS'16
8. Cheng, P., Min, M.R., Shen, D., Malon, C., Zhang, Y., Li, Y., Carin, L.: Improving disentangled text representation learning with information-theoretic guidance. In: Proceedings of the ACL'20
9. Chiang, R.H., Huang Cecil, C.E., Lim, E.P.: Linear correlation discovery in databases: a data mining approach. Data Knowl. Eng. **53**(3), 311–337 (2005)
10. Das, A., Kempe, D.: Algorithms for subset selection in linear regression. In: Proceedings of the STOC'08
11. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of the SCG'04
12. d'Hondt, J., Papapetrou, O., Minartz, K.: Efficient detection of multivariate correlations with different correlation measures. Technical Reports (2023). Available in https://github.com/CorrelationDetective/public
13. Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., Keogh, E.: Querying and mining of time series data: experimental comparison of representations and distance measures. In: Proceedings of the VLDB'08
14. Echihabi, K., Tsandilas, T., Gogolou, A., Bezerianos, A., Palpanas, T.: Pros: data series progressive k-nn similarity search and classification with probabilistic quality guarantees. VLDB J. **32**, 763–789 (2023)
15. Echihabi, K., Zoumpatianos, K., Palpanas, T., Benbrahim, H.: The Lernaean hydra of data series similarity search: an experimental evaluation of the state of the art. In: Proceedings of the VLDB'18
16. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. J. Comput. Syst. Sci. **66**(4), 614–656 (2003)
17. Foundation, S.: SPARK for autism. https://sparkforautism.org/portal/page/autism-research/
18. Garner, W.R.: Uncertainty and Structure as Psychological Concepts. Wiley, New York (1962)
19. Gedik, B., Bordawekar, R.R., Yu, P.S.: Cell Join: a parallel stream join operator for the cell processor. VLDB J. **18**, 501–519 (2009)
20. Handwerker, D.A., Roopchansingh, V., Gonzalez-Castillo, J., Bandettini, P.A.: Periodic changes in fMRI connectivity. Neuroimage **63**(3), 1712–1719 (2012)
21. He, Y., Ganjam, K., Chu, X.: Sema-join: joining semantically-related tables using big table corpora. In: Proceedings of the VLDB'15
22. Heunis, S., Lamerichs, R., Zinger, S., Caballero-Gaudes, C., Jansen, J.F., Aldenkamp, B., Breeuwer, M.: Quality and denoising in real-time functional magnetic resonance imaging neurofeedback: a methods review. Hum. Brain Mapp. **41**(12), 3439–3467 (2020)
23. Härdle, W.K.: Applied Multivariate Statistical Analysis, 2nd edn. Springer, Berlin (2007)
24. Jiang, L., Kawashima, H., Tatebe, O.: Incremental window aggregates over array database. In: Proceedings of the IEEE BigData 2014
25. Kistler, R., Kalnay, E., Collins, W., Saha, S., White, G., Woollen, J., Chelliah, M., Ebisuzaki, W., Kanamitsu, M., Kousky, V., van den Dool, H.: The NCEP/NCAR 50-year reanalysis: monthly means CD-ROM and documentation. Bull. Am. Meteorol. Soc. **82**, 247–268 (2001)
26. Kraskov, A., Grassberger, P.: Mic: mutual information based hierarchical clustering. Information theory and statistical learning, pp. 101–123 (2009)
27. Li, M., Chen, X., Li, X., Ma, B., Vitányi, P.M.B.: The similarity metric. IEEE Trans. Inf. Theory **50**(12), 3250–3264 (2004)
28. Licher, S., Ahmad, S., Karamujić-Čomić, H., Voortman, T., Leening, M.J.G., Ikram, M.A., Ikram, M.K.: Genetic predisposition, modifiable-risk-factor profile and long-term dementia risk in the general population. Nat. Med. **25**(9), 1364–1369 (2019)
29. Liess, S., Agrawal, S., Chatterjee, S., Kumar, V.: A teleconnection between the west Siberian plain and the ENSO region. J. Clim. **30**(1), 301–315 (2017)
30. Mangram, M.E.: A simplified perspective of the Markowitz portfolio theory. Glob. J. Bus. Res. **7**(1), 59–70 (2013)
31. Megumi, F., Yamashita, A., Kawato, M., Imamizu, H.: Functional MRI neurofeedback training on connectivity between two regions induces long-lasting changes in intrinsic functional network. Front. Hum. Neurosci. **9**, 160 (2015)
32. Mitra, I., Lavillaureix, A., Yeh, E., Traglia, M., Tsang, K., Bearden, C.E., Rauen, K.A., Weiss, L.A.: Reverse pathway genetic approach identifies epistasis in autism spectrum disorders. PLoS Genet. **13**, 1–27 (2017)
33. Mueen, A.: Enumeration of time series motifs of all lengths. In: Proceedings of the ICDM'13
34. Mueen, A., Nath, S., Liu, J.: Fast approximate correlation for massive time-series data. In: Proceedings of the SIGMOD'10
35. Nguyen, H.V., Müller, E., Andritsos, P., Böhm, K.: Detecting correlated columns in relational databases with mixed data types. In: Proceedings of the SSDBM'14
36. Nguyen, H.V., Müller, E., Vreeken, J., Efros, P., Böhm, K.: Multivariate maximal correlation analysis. In: Proceedings of the ICML'14
37. Oceanic, N., Administration, A.: NOAA integrated surface dataset. https://www.ncei.noaa.gov/access/search/dataset-search
38. O'sullivan, A., Sheffrin, S.M.: Economics: Principles in Action. Pearson Prentice Hall, London (2003)
39. Rostoker, C., Wagner, A., Hoos, H.: A parallel workflow for real-time correlation and clustering of high-frequency stock market data. In: Proceedings of the IPDPS'07
40. Satuluri, V., Parthasarathy, S.: Bayesian locality sensitive hashing for fast similarity search. In: Proceedings of the VLDB'12
41. Skoltech computer vision | deep billion-scale indexing. https://sites.skoltech.ru/compvision/noimi/
42. Segaran, T.: Programming Collective Intelligence: Building Smart Web 2.0 Applications. O'Reilly Media, Inc., Sebastopol (2007)
43. Studenỳ, M., Vejnarová, J.: The multi-information function as a tool for measuring stochastic dependence. Learn. Gr. Models **89**, 261–297 (1998)
44. Tan, Z., Jamdagni, A., He, X., Nanda, P., Liu, R.P.: A system for denial-of-service attack detection based on multivariate correlation analysis. IEEE Trans. Parallel Distrib. Syst. **25**(2), 447–456 (2014)
45. Wang, J., Zhu, Y., Li, S., Wan, D., Zhang, P.: Multivariate time series similarity searching. Sci. World J. **2014**(1) (2014)
46. Watanabe, S.: Information theoretical analysis of multivariate correlation. IBM J. Res. Dev. **4**(1), 66–82 (1960)
47. Wu, Y., Yu, J., Tian, Y., Sidle, R., Barber, R.: Designing succinct secondary indexing mechanism by exploiting column correlations. In: Proceedings of the SIGMOD'19
48. Yang, K., Shahabi, C.: A PCA-based similarity measure for multivariate time series. In: Proceedings of the ACM-MMDB'04
49. Yang, K., Shahabi, C.: An efficient k nearest neighbor search for multivariate time series. Inf. Comput. **205**(1), 65–98 (2007)
50. Yu, C., Luo, L., Chan, L.L.H., Rakthanmanon, T., Nutanong, S.: A fast LSH-based similarity search method for multivariate time series. Inf. Sci. **476**, 337–356 (2019)
51. Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., Stoica, I.: Discretized streams: fault-tolerant streaming computation at scale. In: Proceedings of the SOSP'13
52. Zhang, X., Pan, F., Wang, W., Nobel, A.: Mining non-redundant high order correlations in binary data. In: Proceedings of the VLDB'08
53. Zhu, Y., Shasha, D.: Statstream: statistical monitoring of thousands of data streams in real time. In: Proceedings of the VLDB'02

54. Zilverstand, A., Sorger, B., Zimmermann, J., Kaas, A., Goebel, R.: Windowed correlation: a suitable tool for providing dynamic fmri-based functional connectivity neurofeedback on task difficulty. PLoS ONE **9**(1), 1-13 (2014)