Multivariate Correlations Discovery in Static and Streaming Data

Koen Minartz
Eindhoven University of Technology
k.minartz@tue.nl

Jens E. d'Hondt
Eindhoven University of Technology
j.e.d.hondt@tue.nl

Odysseas Papapetrou
Eindhoven University of Technology
o.papapetrou@tue.nl

ABSTRACT

Correlation analysis is an invaluable tool in many domains, for better understanding 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 to 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 two 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.

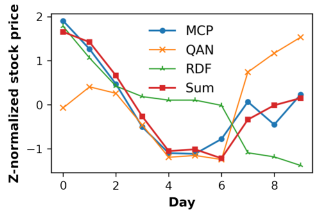
PVLDB Reference Format:
Koen Minartz, Jens E. d'Hondt, and Odysseas Papapetrou. Multivariate Correlations Discovery in Static and Streaming Data. PVLDB, 15(6): 1266-1278, 2022.
doi:10.14778/3514061.3514072

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/CorrelationDetective/public.

1 INTRODUCTION

Correlation analysis is one of the key tools in the arsenal of data analysts for exploring 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 [11]. In finance, correlation plays a crucial role in finding portfolios of assets that are on the Pareto-optimal frontier of risk and expected returns [16], and in genetics, correlations help scientists detect cause factors for hereditary syndromes.[1] Correlations – as a generalization of functional dependencies – also found use for optimizing access paths in databases [29].

Multivariate correlations, or high-order correlations, are a generalization of pairwise correlations that can capture relations among arbitrarily-sized sets of variables, represented as high-dimensional vectors or as time series.[2] Multivariate correlations have 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 tasks [1, 2], and in climatology, a ternary correlation led to the characterization of a new weather phenomenon and to improved climate models [15]. Furthermore, a more thorough look at multivariate correlations may open doors in the fields of genomics [23, 30] and medicine [14, 19].



	MCP	QAN	RDF	Sum
MCP	1	0.46	0.53	0.96
QAN	0.46	1	-0.47	0.52
RDF	0.53	-0.47	1	0.52
Sum	0.96	0.52	0.52	1

Figure 1: (a) Normalized daily closing prices for stocks traded at the Australian Securities Exchange, (b) Correlation matrix of the prices.

Accordingly, several measures and algorithms for discovering multivariate correlations have been proposed, such as tripoles [1], multipoles [2], Canonical Correlation Analysis [13] and Total Correlation (TC) [28] and its variants [21, 22, 30]. However, the proposed algorithms do not sufficiently address the fundamental impediment on the discovery of strong multivariate correlations, which is the vast search space. Unfortunately, apriori-like pruning techniques do not apply for the general case of multivariate correlations. For example, consider the three time series presented in Fig. 1, which represent closing prices of three stocks from the Australian securities exchange. In this case, the pairwise correlations between all pairs of the three time series are comparatively low, whereas the time series created by summing QAN and RDF is strongly correlated to MCP. 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 among 1000 vectors involves 1 trillion combinations. The mere generation and enumeration of these combinations already becomes challenging. Therefore, smart algorithms are needed that can prune the search space to reduce computational complexity.

Existing algorithms (see Section 2.3) follow at least one of the following approaches: (a) they consider constraining definitions of multivariate correlations that enable apriori-like filtering [2, 21, 30], (b) they rely on hand-crafted additional assumptions of the

Proceedings of the VLDB Endowment, Vol. 15, No. 6 ISSN 2150-8097.
doi:10.14778/3514061.3514072

[1]A prime example is the Spark project for discovering gene properties related to the manifestation of the autism spectrum disorder [9], which led to a list of genes and their correlated symptoms [10].

[2]In the remainder of this paper we will generally refer to the more general case of vectors, but often the data consists of time series that may come with live updates.

user query, which may be too constraining for other application scenarios [1, 2, 30], or, (c) they offer approximate results, with no guarantees [1, 2]. Even though these algorithms are relevant for their particular use cases, they are not generally applicable.

In this work, we follow a more general direction. First, we also consider correlation measures that are unsuitable for apriori-like pruning. Although their usefulness has already been validated in multiple use cases, e.g., [1, 2, 15], algorithms for detecting them do not scale. Second, 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. Finally, we extend the proposed methods to a dynamic context by efficiently handling streaming data, enabling use-cases where continuous updates of query answers are required, such as flash-trading models in finance [25], weather and server monitoring [27], and neurofeedback training [12, 17, 32].

We evaluate our algorithms on 3 datasets and compare them to the state-of-the-art. Our evaluation demonstrates that we outperform the existing methods by typically an order of magnitude, and the exhaustive-search baseline by several orders of magnitude. Finally, we show that the progressive version of the algorithm produces around 80% 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 (Section 3), and the streaming extension of the algorithm (Section 4). Section 5 summarizes the experimental results. We conclude the paper in Section 6.

2 PRELIMINARIES

We start with a discussion of the multivariate correlation measures that will be considered in this work. We then formalize the problem, and discuss prior work.

2.1 Correlation measures

Our work focuses on two multivariate correlation measures: the two-sided multiple correlation, and the one-sided multipole.

Multiple correlation. Given two sets of vectors X and Y, multiple correlation is defined as follows:

$$\mathbf{mc}(X,Y) = \rho\left(\frac{\sum_{\mathbf{x}\in X}\hat{\mathbf{x}}}{|X|}, \frac{\sum_{\mathbf{y}\in Y}\hat{\mathbf{y}}}{|Y|}\right) \tag{1}$$

where ρ denotes the Pearson correlation coefficient and $\hat{\mathbf{x}}$ denotes $\mathbf{x}$ after z-normalization, i.e., $\hat{\mathbf{x}}_i = \frac{\mathbf{x}_i - \mu_{\mathbf{x}}}{\sigma_{\mathbf{x}}}$. Both the definition and this work can be easily extended to weighted linear aggregates, instead of averaging. Tripoles [1] is a special case of the multiple correlation measure, where $|X| = 2$ and $|Y| = 1$. In this work, we allow both X and Y to contain more vectors.

Multipole. The multipole correlation $\mathbf{mp}(X)$ measures the linear dependence of an input set of vectors X [2]. 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:

$$\mathbf{mp}(X) = 1 - \min_{||\mathbf{v}||_2=1} \mathrm{var}(\mathbf{X}\cdot\mathbf{v}) \tag{2}$$

The value of $\mathbf{mp}(X)$ lies between 0 and 1. The measure takes its maximum value when there exists perfect linear dependence, i.e., there exists a vector $\mathbf{v}$ with norm 1, such that $\mathrm{var}(\mathbf{X}\cdot\mathbf{v}) = 0$.

Notice that multipoles is not equivalent to, nor a generalization of, multiple correlation. By definition, $\mathbf{mp}$ assumes optimal weights (vector $\mathbf{v}$ is such that the variance is minimized), whereas for $\mathbf{mc}$, the linear aggregation function for the vectors is determined at the definition of the measure. Furthermore, $\mathbf{mp}(\cdot)$ expresses the degree of linear dependence within a single set of vectors, whereas for $\mathbf{mc}(\cdot,\cdot)$, two distinct, non-overlapping vector sets are considered.

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 as $Corr(X,Y)$, with the understanding that for the definitions of $Corr$ that expect one input, 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 $l_{\max}, r_{\max} \in \mathbb{N}$, find all pairs of sets ($X \subset \mathcal{V}, Y \subset \mathcal{V}$), for which $Corr(X,Y) \geq \tau$, $X \cap Y = \emptyset$, $|X| \leq l_{\max}$ and $|Y| \leq r_{\max}$.

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

The combination of $l_{\max}$ and $r_{\max}$ controls the desired complexity of the answers. Smaller $l_{\max} + r_{\max}$ 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 [30]) can easily be integrated in the algorithm:

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 $(X',Y') \neq (X,Y)$. Intuitively, if $Corr(X',Y') \geq \tau$, then no supersets of X' and Y' should be considered together. This constraint prioritizes simpler and more interpretable answers.

Minimum jump constraint: For each (X,Y) in the result set, there exists no (X',Y') such that $X' \subseteq X$, $Y' \subseteq Y$, $(X',Y') \neq (X,Y)$, and $Corr(X,Y) - Corr(X',Y') < \delta$. This constraint, which was first proposed in [1], discards solutions where a vector in $X \cup Y$ contributes less than δ to the increase of the correlation.

The minimum jump constraint applies to both query types, whereas the irreducibility constraint is only useful for threshold queries. For top-κ queries, irreducibility is ill-defined: 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 denote the combination of the correlation measure, $l_{\max}$ and $r_{\max}$ as $\mathbf{mc}(l_{\max}, r_{\max})$ (for $\mathbf{mc}$) and $\mathbf{mp}(l_{\max})$ (for $\mathbf{mp}$). We will call this a *correlation pattern*. For example, $\mathbf{mc}(2,1)$ will identify the combinations of sets of vectors of size 2 and 1 with high $\mathbf{mc}$ correlation. Pattern $\mathbf{mp}(4)$ will identify

Table 1: Properties of the most relevant related work for multivariate correlations, and the proposed method.

	Complete	Require constraints	Correlation Measures	Query types
[1]	Yes	Yes	$\mathbf{mc}(1, 2)$	Threshold
[2]	No	Yes	$\mathbf{mp}(\cdot)$	Threshold
[21]	No	No	$TC(\cdot)$	Threshold
[30]	Yes	Yes	$TC(\cdot)$ (only binary data)	Threshold
Ours	Yes	No	$\mathbf{mc}(\cdot, \cdot)$, $\mathbf{mp}(\cdot)$	Threshold, top-κ

the combinations of vectors of size at most 4 with high multipoles correlation. Finally, we will denote a particular combination of vectors – a *materialization* of the correlation pattern – by displaying the vectors, grouped by parentheses. For example, $(\mathbf{v}_1, (\mathbf{v}_2, \mathbf{v}_3))$ denotes a combination for the multiple correlation measure, where vectors $\mathbf{v}_2$ and $\mathbf{v}_3$ are aggregated together.

2.3 Related work

Several algorithms exist for efficiently finding highly correlated *pairs* in large sets of high-dimensional vectors, e.g., time series. For example, StatStream [31] and Mueen et al. [20] map pairwise correlations to Euclidean distances, and exploit Discrete Fourier Transforms, grid-based indexing, and dynamic programming to reduce the search space. Other works proposing indices for high dimensional Euclidean data [7, 26] are applicable as well due to the one-to-one mapping of Pearson correlation to Euclidean distance. 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., example of Fig. 1).

Agrawal et al. [1] investigate the problem of finding highly-correlated tripoles – a special case of **mc** that contains exactly three vectors. Their algorithm relies on the minimum jump constraint for effective pruning. Compared to tripoles, our work handles the more general definition of multiple correlation, allowing more vectors at the left and right hand side. Moreover, our work does not require the use of the minimum jump constraint to prune comparisons.

Algorithms for discovering high correlations according to the multipole measure (Eqn. 2) were proposed in [2]. Both CoMEt and CoMEtExtended are approximate algorithms relying on clique enumeration and the minimum jump constraint to efficiently explore the search space. Their efficiency depends on a parameter ρ_{CE} that trades off completeness of the result set for performance. Both algorithms yield more complete result sets compared to methods based on l_1-regularization and structure learning. Still, they do not offer completeness guarantees. In contrast, our work is exact – it always retrieves all answers – and outperforms both algorithms.

Total correlation is an non-linear information-theoretic metric that expresses how much information is shared between variables [28]. Nguyen et al. [21] proposed a closely related correlation measure, and an algorithm for finding strongly correlated groups of columns in a database. The key idea of their method is to first evaluate all pairwise correlations, and use those to calculate a lower bound on the total correlation of a group. Their algorithm subsequently finds quasi-cliques in which most pairwise correlations are high, implying a high total correlation value. However, groups with low pairwise correlations can still be strongly correlated as a whole, and these are arguably the most interesting cases. As such, the method is effectively an approximation algorithm. In another work, Zhang et al. also developed an algorithm that discovers sets with a high total correlation value [30]. However, the method is limited to data with binary features, and relies on a limiting weak-correlated subset constraint.

In the supervised learning context, subset regression appears similar to multivariate correlation mining. The goal of this feature selection problem is to select the best p predictors out of n features [6]. Our problem differs from the above in that we aim to find interesting patterns in the data, instead of finding the best predictors for a *given* dependent variable. Further, instead of finding only the single highest correlated set of vectors, our goal is to find a *diverse set* of results, enabling the domain expert to assess a variety of results on qualitative aspects and to gain more insights.

Table 1 summarizes the properties of the most closely related work.

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 combinations that need to be examined. In a dataset of n vectors, there exist at least $O\left(\sum_{p=2}^{l_{\max}+r_{\max}} \binom{n}{p}\right)$ possible combinations. Even if each possible combination can be checked in constant time, their enumeration still requires significant computational effort.

Our algorithm – Correlation Detective, abbreviated as *CD* – exploits the insight that vectors often exhibit (possibly weak) correlations between each other. For example, securities that relate to the same conglomeration (e.g., Fig. 2(a), 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.

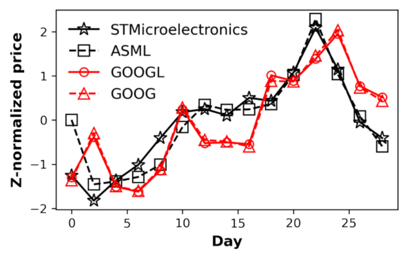
CD works as follows: rather than iterating over all possible vector combinations that correspond to the correlation pattern, CD clusters vectors, and enumerates the combinations of only the cluster centroids. For each of these combinations, it computes an upper and lower bound on the correlations of 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.

In the remainder of this section, we will present the algorithm and explain how the two types of queries presented in Section 2 are handled. We will start with a brief description of the initialization and clustering phase. In Sections 3.2 and 3.3 we will describe how CD answers threshold and top-κ queries respectively.

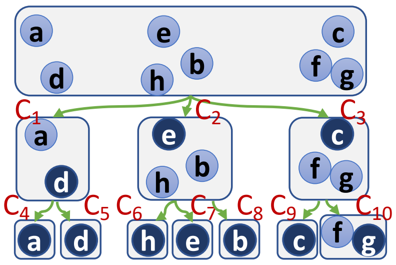
3.1 Initialization and clustering

First, all vectors are z-normalized, i.e., shifted and scaled such that they have zero mean and unit standard deviation. From here on, the algorithm operates only on z-normalized vectors.

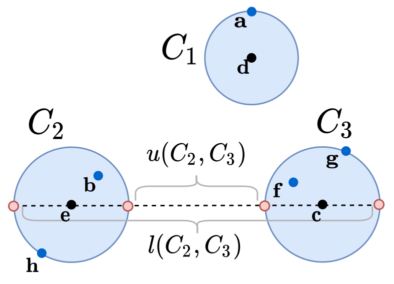
Next, we hierarchically cluster all vectors. The clustering algorithm operates in top-down fashion. A root cluster containing all



(a)



(b)



(c)

Figure 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 (schematic): 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 3.1.

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++ [3]. The use of K-means++ (as opposed to sampling K random vectors) ensures that these initial centroids are well-distributed over the Euclidean space. In the second step, we run standard K-means 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 initial 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 results in a higher clustering quality (lower WCSS), but takes longer to compute. However, clustering quality does *not* affect the correctness of CD: regardless of the clustering algorithm, configuration, or final solution, CD always returns the correct results. Poor clustering can only affect 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 were chosen) 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 poorly chosen initial centroids, we set $r_2 = 50$. Still, 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, a correlation function *Corr*, 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. 2(b), for a desired correlation pattern of $\mathbf{mc}(2, 1)$, the following *combinations of clusters* are examined in the

```
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, 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        $\left(\mathcal{S}'_l, \mathcal{S}'_r\right) \leftarrow (\mathcal{S}_l, \mathcal{S}_r)$ with $C_{max}$ replaced by $S$
11        ThresholdQuery$\left(\left(\mathcal{S}'_l, \mathcal{S}'_r\right), Corr, \tau\right)$
```

order of increasing pattern length:
$\forall_{C_x, C_y \in \{C_1, C_2, C_3\}} (C_x, C_y) \cup \forall_{C_x, C_y, C_z \in \{C_1, C_2, C_3\}} ((C_x, C_y), C_z)$

A combination of clusters compactly represents the combinations created by the Cartesian product of the vectors inside the clusters. For each such combination, the algorithm computes lower and upper bounds on the correlation of these clusters, denoted with LB and UB respectively (Alg. 1, line 1). These bounds, derived later in this section, 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 $LB \geq \tau$, the combination is *decisive positive*, guaranteeing that all possible materializations of this combination will have a correlation of at least τ. Therefore, all materializations are inserted in the result. If $UB < \tau$, the combination is *decisive negative* – no materialization yields a correlation higher than the threshold τ. Therefore, this combination does not need to be examined further. 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, and recursively checks all combinations where $C_{\max}$ is replaced by one of its sub-clusters. In the example of Figure 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 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. Decisive combinations are typically found at high levels of the cluster tree, thereby saving many comparisons.

In the 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 cardinality of the clusters. 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 Pearson correlation between only two clusters, which serves as a stepping stone for multivariate correlations.

Lemma 3.1. *Let $\rho(\mathbf{x}, \mathbf{y})$ denote the Pearson correlation between two vectors $\mathbf{x}$ and $\mathbf{y}$, and $\theta_{\mathbf{x},\mathbf{y}}$ the angle formed by these vectors. Consider four z-normalized 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, correlation $\rho(\mathbf{u_1}, \mathbf{u_2})$ can be bounded as follows:*

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

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 the technical report [18]. □

Lemma 3.1 bounds the correlation 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. 2(b)), we can bound the correlation between any two vectors from (C_1, C_2) if we have the cosine of the two cluster centroids $\mathbf{d}$ and $\mathbf{e}$, the cosines of $\mathbf{a}$ with $\mathbf{d}$, and $\mathbf{h}$ with $\mathbf{e}$ (as $\mathbf{h}$ is the furthest point in C_2 from the centroid $\mathbf{e}$). The bounds are tightened if the maximum angle formed by each centroid with all cluster vectors is reduced.

We now extend our discussion to cover multivariate correlations, which involve three or more clusters. We first derive bounds for **mc** (Theorem 3.2), and then for **mp** (Theorem 3.3).

Theorem 3.2 (Bounds for **mc**). *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 clusters' 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 $\mathcal{S} = \{C_1, C_2, \ldots, C_N\}$, partitioned into $\mathcal{S}_l = \{C_i\}_{i=1}^{l_{max}}$ and $\mathcal{S}_r = \{C_i\}_{l_{max}+1}^{N}$. 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 $X_l = \{\mathbf{x_1}, \ldots, \mathbf{x_{l_{max}}}\}$, $X_r = \{\mathbf{x_{l_{max}+1}}, \ldots, \mathbf{x_N}\}$ such that $\mathbf{x_i} \in C_i$, multiple correlation $\mathbf{mc}(X_l, X_r)$, can be bounded as follows:*

(1) *if $L(\mathcal{S}_l, \mathcal{S}_r) \geq 0$:*
$$\frac{L(\mathcal{S}_l, \mathcal{S}_r)}{\sqrt{U(\mathcal{S}_l, \mathcal{S}_l)}\sqrt{U(\mathcal{S}_r, \mathcal{S}_r)}} \leq \mathbf{mc}(X_l, X_r) \leq \frac{U(\mathcal{S}_l, \mathcal{S}_r)}{\sqrt{L(\mathcal{S}_l, \mathcal{S}_l)}\sqrt{L(\mathcal{S}_r, \mathcal{S}_r)}}$$

(2) *if $U(\mathcal{S}_l, \mathcal{S}_r) \leq 0$:*
$$\frac{L(\mathcal{S}_l, \mathcal{S}_r)}{\sqrt{L(\mathcal{S}_l, \mathcal{S}_l)}\sqrt{L(\mathcal{S}_r, \mathcal{S}_r)}} \leq \mathbf{mc}(X_l, X_r) \leq \frac{U(\mathcal{S}_l, \mathcal{S}_r)}{\sqrt{U(\mathcal{S}_l, \mathcal{S}_l)}\sqrt{U(\mathcal{S}_r, \mathcal{S}_r)}}$$

(3) *else:*
$$\frac{L(\mathcal{S}_l, \mathcal{S}_r)}{\sqrt{L(\mathcal{S}_l, \mathcal{S}_l)}\sqrt{L(\mathcal{S}_r, \mathcal{S}_r)}} \leq \mathbf{mc}(X_l, X_r) \leq \frac{U(\mathcal{S}_l, \mathcal{S}_r)}{\sqrt{L(\mathcal{S}_l, \mathcal{S}_l)}\sqrt{L(\mathcal{S}_r, \mathcal{S}_r)}}$$

Combined with Lemma 3.1, Theorem 3.2 enables bounding the multiple correlation of any cluster combination that satisfies the 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 3.1 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 $\mathbf{mc}((C_1, C_2), C_3)$ then follow directly from Theorem 3.2.

Also, observe that by tightening the bounds for the pairwise correlations, we can tighten $L(\cdot,\cdot)$ and $U(\cdot,\cdot)$, which will in turn tighten the bounds for **mc**. This is further exploited in Section 3.2.2.

Theorem 3.3 (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 $\mathcal{S} = \left\{C_1, C_2, \ldots, C_{l_{max}}\right\}$. Furthermore, let $\mathbf{L}$ and $\mathbf{U}$ be symmetric matrices with elements $l_{ij} = l(C_i, C_j)$ and $u_{ij} = u(C_i, C_j)$ $\forall 1 \leq i, j \leq l_{max}$. For any set of vectors $X = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_{l_{max}}}\}$ such that $\mathbf{x_i} \in C_i$, multipole correlation $\mathbf{mp}(X)$ can be bounded as follows:*

$$1 - \lambda_{min} - \frac{1}{2}||\mathbf{U} - \mathbf{L}||_2 \leq \mathbf{mp}(X) \leq 1 - \lambda_{min} + \frac{1}{2}||\mathbf{U} - \mathbf{L}||_2$$

where λ_{min} is the smallest eigenvalue of matrix $\frac{\mathbf{L}+\mathbf{U}}{2}$. □

Similar to Theorem 3.2 for **mc**, the tightness of the bounds from Theorem 3.3 depend on the tightness of the bounds for the pairwise correlations between clusters, which can be derived with Lemma 3.1. Proofs for both theorems can be found in [18].

3.2.2 Empirical pairwise bounds. The bounds of Lemma 3.1 – which determine the bounds of Theorems 3.2 and 3.3 – tend to be pessimistic, as they always account for the worst case. In the example of Fig. 2(c), the theoretical lower bound (resp. upper bound) accounts for the case that hypothetical vectors (depicted in pink) are located on the clusters' edges such that they are as far away from (resp. as close to) each other as possible, given the position of the cluster centroids (depicted in black) and cluster radii.

The *empirical bounds* approach builds on the observation that the pairwise correlations 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 correlations and store these in an upper-triangular matrix. Note that part of these correlations have already been calculated during the clustering phase. Then, during execution of Alg. 1, we lazily compute $l(C_i, C_j)$ and $u(C_i, C_j)$ as follows: $l(C_i, C_j) = \min_{\mathbf{x}\in C_i, \mathbf{y}\in C_j} \rho(\mathbf{x}, \mathbf{y})$ and $u(C_i, C_j) = \max_{\mathbf{x}\in C_i, \mathbf{y}\in C_j} \rho(\mathbf{x}, \mathbf{y})$, with $\rho(\mathbf{x}, \mathbf{y})$ 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 3.2 and 3.3 that $l(C_i, C_j) \leq \min_{\mathbf{x}\in C_i, \mathbf{y}\in C_j} \rho(\mathbf{x}, \mathbf{y})$ and $u(C_i, C_j) \geq \max_{\mathbf{x}\in C_i, \mathbf{y}\in C_j} \rho(\mathbf{x}, \mathbf{y})$. Consequently, bounds on **mc** and **mp** derived using empirical bounds are still correct. Moreover, they are at least as tight as the bounds of Lemma 3.1, 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 3.1 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 3.2 by 50% to 90% (for $\mathbf{mc}(1, 2)$), which empowered CD to reach to decisive combinations faster. As a result, total execution time of CD 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.

3.2.3 Exploiting additional constraints. CD supports both the irreducibility and minimum jump constraints (see Section 2.2). 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 Alg. 1, only the pairwise correlations between any two clusters $C_l \in \mathcal{S}_l$ and $C_r \in \mathcal{S}_r$ are examined (for **mp**, both $C_l \in \mathcal{S}_l$ and $C_r \in \mathcal{S}_l$). Precisely, we use $l(C_l, C_r)$, which is already computed for Theorems 3.2 and 3.3. 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 Alg. 1, then the combination is discarded. However, considering only the pairwise correlations during the pruning process may lead to inclusion of answers that do not satisfy the constraints. Therefore, 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.

3.3 Top-κ queries

When exploring new datasets, it may be difficult to decide on a threshold τ. Setting the threshold too high for the dataset may lead to no answers, whereas a very low τ can result in millions of answers, and performance decrease. 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 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 [8]) 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.

```
Algorithm 2: TOP-κ-QUERY($\mathcal{S}_l$, $\mathcal{S}_r$, $Corr$, $\tau$, $\kappa$, $\gamma$, $B$)
  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 $\gamma$, list of buckets $B$.
1  $(LB, UB_{shrunk}) \leftarrow$ CALCBOUNDS($\mathcal{S}_l, \mathcal{S}_r, Corr, \gamma$)
2  if $LB \geq \tau$ then
3      Add the contents of $(\mathcal{S}_l, \mathcal{S}_r)$ to the result set $\mathcal{R}$
4      $\mathcal{R} \leftarrow$ SORT($\mathcal{R}$)[1:$\kappa$]
5      $\tau \leftarrow \min_{(X,Y)\in\mathcal{R}} Corr(X, Y)$
6  else if $UB_{shrunk} \geq \tau$ then
       // Replace largest cluster with subclusters and recurse
       with TOP-κ-QUERY (similar to lines 7-11 of Alg. 1)
12 else
13     $\gamma^* = \frac{\tau-\mu}{UB-\mu}$
14     Assign $(\mathcal{S}_l, \mathcal{S}_r)$ to bucket $\lceil \gamma^* \cdot |B| \rceil$
   // Phase 2 - starts when Phase 1 is completed
15 for $b \in B$ do
16     for $(\mathcal{S}_l, \mathcal{S}_r) \in b$ do
17         THRESHOLDQUERY($\mathcal{S}_l$, $\mathcal{S}_r$, $Corr$, $\tau$)
18     $\mathcal{R} \leftarrow$ SORT($\mathcal{R}$)[1:$\kappa$]
19     $\tau \leftarrow \min_{(X,Y)\in\mathcal{R}} Corr(X, Y)$
```

The top-κ variant of CD (see Alg. 2) follows the same idea. The algorithm has the same core as the threshold-based variant, and relies on two orthogonal techniques to increase τ quickly. First, at invocation, 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.

The second 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 Alg. 1, the algorithm computes the upper and lower bound per combination. However, it now artificially tightens the bounds by moving the upper bound towards the lower bound. This so-called *shrinking* is achieved by taking $UB_{\text{shrunk}} = (1-\gamma)\cdot\mu+\gamma\cdot UB$, where $\mu = \frac{UB+LB}{2}$ and $\gamma \in [0, 1]$ is a shrink factor with a default value of 0. If the lower bound surpasses the current threshold τ, all solutions resulting from this candidate combination are added to the set of answers $\mathcal{R}$, and the κ solutions from $\mathcal{R}$ with the highest correlation are kept (Alg. 2, lines 3-4). The value of τ is then set to the minimum correlation in $\mathcal{R}$ (line 5). Otherwise, if UB_{shrunk} is greater than the running τ, we recursively break the cluster to smaller clusters, until we get decisive bounds, analogous to Alg. 1 (lines 6-11). Finally, if the shrunk upper bound is less than the running value of τ but the true UB is greater than τ, we compute the critical shrink factor γ^* for the cluster (line 13) – the minimum value of γ for which UB_{shrunk} would surpass τ. Intuitively, a small γ^* means that the combination is more promising to lead to higher correlation values.

All combinations are placed in B equi-width buckets based on their γ^* values (line 14). At the second phase (lines 15-19), the algorithm processes the buckets one by one, starting from the first, invoking the threshold query algorithm on each of its cluster combinations (Alg. 1) and updating the running τ after every bucket. Since τ continuously increases, and the first buckets are likely to contain the highest correlation values, most combinations after the first few buckets will be filtered without needing many cluster splits.

3.3.1 Progressive threshold queries. The prioritization technique of Alg. 2 can also be used as a basis for a progressive threshold algorithm. Precisely, Alg. 2 can be initialized with a user-chosen τ and with $\kappa \rightarrow \infty$. 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 Alg. 1, or interrupt the algorithm after receiving sufficient answers. We will evaluate the progressive nature of CD in Section 5.

4 DETECTION OF MULTIVARIATE CORRELATIONS IN STREAMING DATA

Our streaming algorithm, called CDStream, builds on top of CD such that it maintains CD's solution over a sliding window as new data arrive. Currently, CDStream works with the multiple correlation measure only; efficient support for the multipole measure is ongoing work.

CDStream relies on two observations to increase the performance for streaming data. First, most arrivals do not lead to significant updates to the final result. Second, in most real-world scenarios, each of the streams may have a different update rate. For example, in finance, each stock exchange serves updates at different frequencies. The one-size-fits-all approach of CD that handles all updates identically, recomputing the full solution from scratch can be wasteful.

At initialization, CDStream executes CD on the initial data. Then, the core idea of CDStream is as follows. Assume an update of a vector $\mathbf{v}$. This vector belongs to a hierarchy of clusters. For example, vector $\mathbf{e}$ in Fig. 2(b) belongs to C_2 and C_7. We denote the set of these clusters as $\mathcal{C}(\mathbf{v})$. The cluster combinations that need to be checked after the update of $\mathbf{v}$ are only the decisive combinations – either positive or negative – that involve a cluster from $\mathcal{C}(\mathbf{v})$. The final result remains correct if these combinations are still decisive positive/negative.

To understand how CDStream adds pruning power on top of CD, observe that even for combinations with three or more clusters, the combination's bounds are determined by $l(C_i, C_j)$ and $u(C_i, C_j)$, the minimum and maximum *pairwise* correlations between all involved clusters. Therefore, any update that does not change $l(C_i, C_j)$ and $u(C_i, C_j)$ for all pairs of involved clusters cannot invalidate the previous bounds, or the previous solution. We refer to the pairs of vectors from C_i and C_j that are responsible for $l(C_i, C_j)$ and $u(C_i, C_j)$ as the **minimum** and **maximum extrema pair** respectively. For example, in Fig. 2(c), the minimum and maximum extrema pairs for (C_2, C_3) are $\langle \mathbf{h}, \mathbf{g} \rangle$ and $\langle \mathbf{b}, \mathbf{f} \rangle$ respectively. CDStream exploits this observation by: (a) checking if each update causes a change to any $l(C_i, C_j)$ and $u(C_i, C_j)$, and (b) for the updates that indeed cause a change, updating the extrema pairs, and recomputing the bounds with Theorem 3.2 and the final solution.

Key to the performance of CDStream is an index that enables the algorithm to quickly locate the extrema pairs that are potentially affected by an update. The index maps each vector $\mathbf{v}$ to a list of all decisive combinations (both negative and positive) that involve any cluster from $\mathcal{C}(\mathbf{v})$. Internally, the combinations for $\mathbf{v}$ are grouped in two levels. First, they are grouped by the extrema pairs. For example, in Fig. 3, the first 5 combinations for vector $\mathbf{c}$ are grouped under extrema pair $\langle \mathbf{b}, \mathbf{f} \rangle$. All combinations with the same extrema pair are subsequently grouped by the cluster that does not contain the vector used for indexing (in this case, vector $\mathbf{c}$). In our example, the first two combinations have C_2 as the second cluster and are grouped together. We will refer to these clusters for the same extrema pair as the **extrema pair clusters**. The described index is constructed at initialization by iterating over all vectors V in a decisive combination when it is identified (i.e., Alg. 1 lines 3,5), and storing it in the index on every extrema pairs that a vector $\mathbf{v} \in V$ can violate (see Alg. 5 of [18] for pseudocode).

The index is used to support a triggering functionality, which allows us to quickly locate and verify the extrema pairs related to each update. First, the index is used to retrieve the information related to an updated vector $\mathbf{v}$. The algorithm iterates over the respective extrema pairs to verify that these did not change or move, despite the update (Alg. 3, lines 2-11)[3]. Precisely, for each minimum (resp. maximum) extrema pair with correlation $\rho_{\min}$ ($\rho_{\max}$), it verifies that the correlation of $\mathbf{v}$ with all points belonging to the second cluster is still at least $\rho_{\min}$ (at most $\rho_{\max}$) (line 6). If this is still the case, all decisive combinations are still valid. If, on the other hand, an extrema pair is invalidated the respective decisive combinations are checked and their bounds are recomputed and updated. Combinations are re-indexed in case extrema pairs have changed. In case a combination becomes indecisive, subcluster combinations are checked analogously to Alg. 1 lines 7-11, storing new combinations through the standard indexing procedure described earlier. Indecisive combinations are removed from the index in a lazy manner.

Checking whether $\rho_{\min}$ (resp. $\rho_{\max}$) are still valid requires computing the correlation of $\mathbf{v}$ with each of the vectors contained in all extrema pair clusters. A critical observation is that there always exists one cluster in the extrema pair clusters that contains all others – otherwise the other clusters could not contain the same extrema vector. Therefore, the algorithm considers the extrema pair clusters in decreasing size. If the largest cluster passes the test, then all its decisive combinations and all the decisive combinations of all its sub-clusters are still valid and do not need to be checked (lines 9-11). In the running example, if $\langle \mathbf{b}, \mathbf{f} \rangle$ is still the maximum extrema pair between clusters C_3 and C_2, and it has the same $\rho_{\max}$, then all combinations under $\langle \mathbf{b}, \mathbf{f} \rangle$ are still decisive. If the largest cluster does not pass the test, then the bounds for all its decisive combinations are verified. The combinations that are no longer decisive are updated accordingly, e.g., by breaking one of the involved clusters to sub-clusters, as described in Section 3.2. Furthermore, the second,

[3]Alg. 3 describes the process of querying the DCC Index for validating the maximum extrema pairs. The process of validating the minimum extrema pairs is analogous.

```
Algorithm 3: QueryIndex($i, \mathcal{I}$)
  Input: A stream index $i$, the DCC Index $\mathcal{I}$
  Output: A set of DCCs $O$ that need to be checked
1  $O \leftarrow \{\}$                                  // Initialize output set
2  for $\langle a, b\rangle \in \mathcal{I}[i]$ do          // Iterate over extrema pairs
3      $V \leftarrow \{\}$                     // Vectors violating the extrema
4      $C_p \leftarrow \mathcal{I}[i][\langle a, b\rangle][0]$         // Get largest cluster
5      for $v_j \in C_p$ do    // Iterate over cluster content
6          if $\rho(v_i, v_j)_t > \rho(a, b)_t$ then
7              $V \leftarrow V \cup v_j$                // Add to violations
8              $O \leftarrow O \cup \mathcal{I}[i][\langle a, b\rangle][C_p]$          // Add DCCs
9      for $C \in \mathcal{I}[i][\langle a, b\rangle][1:]$ do   // Check sub-clusters
10         if $C \cap V \neq \emptyset$ then       // Violating point in $C$
11             $O \leftarrow O \cup \mathcal{I}[i][\langle a, b\rangle][C]$          // Add DCCs
12 return $O$
```

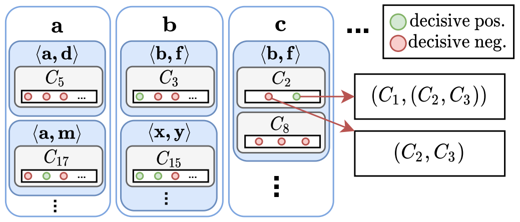


Figure 3: Visualization of the decisive combination index

third, etc. largest extrema pair clusters are tested recursively. The process stops as soon as one of these clusters passes the test.

This grouping of decisive combinations based on the extrema pairs and clusters is instrumental in the algorithm's efficiency, as each pair of clusters may appear in many decisive combinations. In the example of Fig. 2(c), assuming that $l_{\max} + r_{\max} = 3$ with **mc** measure, C_2 and C_3 will appear in a combination of size 2 without C_1, and in a combination of size 3, together with C_1. In both cases, the extrema pairs between C_2 and C_3 will be identical. Therefore, with a single check, both decisive combinations can be verified. Typically the number of decisive combinations for each pair and for each cluster is in the order of a few hundreds for $n = 1000$.

CDStream supports discretization of the stream of updates to small batches (e.g., of a few seconds, or a few tens or hundreds of updates) as a method to trade-off throughput and freshness of results. A larger batch size increases performance and throughput, but potentially delays the updating of the final results. In Section 5 we will evaluate CDStream with different batch sizes.

4.1 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: (a) (X, Y) needs to be removed from the result set because (X', Y') just surpassed τ, and, (b) (X, Y) needs to be added in the result set, because (X', Y') was just removed from the result set. Both cases can be triggered by an update of a vector from X or 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 (a), 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 contraint. The following cases need to be considered: (a) (X, Y) needs to be removed from the result set because $Corr(X', Y') + \delta > Corr(X, Y)$, and (b) (X, Y) needs to be added in the result set because $Corr(X, Y) > \tau$ and $Corr(X', Y') + \delta < Corr(X, Y)$. 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 $\kappa' = b * \kappa$, where b is at least 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 controls the tradeoff between the robustness of the streaming algorithm for top-κ queries, and its efficiency. Setting $b = 1$ 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 resorts to CD for computing the correct answer, and updating its index. Conversely, a large b 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 $b = 2$ is already sufficient to provide good performance without compromising the robustness of CDStream.

4.2 CDHybrid: combining CD and CDStream

Recall that CDStream handles the stream updates in batches. 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 run the one-shot algorithm after the completion of each batch 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 a batch. A good predictor for this is the number of updates in the batch – more updates 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 linear regression is then used to model the relationship between execution time and the observed number of updates. 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 a batch is completed, the vectors are updated (i.e., by progressing the sliding window of the each vector) 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 negative decisive combination, which needs to be added in the index, whereas any r contained in $\Delta \cap \mathcal{R}_{\text{CD}}$ leads to a new positive decisive combination.

Notice that the switch from CD to CDStream will not remove from the index the decisive combinations that were constructed from CDStream, but are no longer relevant, e.g., because CD split one of its involved clusters. We use a lazy approach to detect these combinations in the index: the first time we access a combination after the switch, we check if there exists a result $r \in \Delta$ that is included in the cluster combination. If so, we reconstruct the combination such that r is removed from it. For example if we access (C_1, C_3), and decisive combination (C_1, C_9) is in Δ, we replace $(C_1 C_3)$ with (C_1, C_{10}) and move it to the correct place in the index. If all possible vector combinations in the combination are in Δ, the combination is discarded from the index.

5 EXPERIMENTAL EVALUATION

The purpose of our experiments was twofold: (a) to assess the scalability and efficiency of our methods for varying input parameters, and, (b) to compare them with the state-of-the-art algorithms for multivariate correlation discovery [1, 2], and an exhaustive search baseline that iterates over all possible combinations. The practical significance of multivariate correlations with the two correlation measures was already extensively demonstrated in different domains, e.g., [1, 2, 15] (see Section 1 for more examples). Since CD supports the same correlation measures (and further generalizations of them), and guarantees completeness of results, we do not repeat their use-case studies, but evaluate our methods on the same data (or data of the same type, where the original data was unavailable).

Hardware and implementations. All experiments were executed on a server equipped with a 24-cores Intel Xeon Platinum 8260 Processor, and 400GB RAM. For CoMEtExtended and CONTRa, we used the original implementations, which were kindly provided by the authors [1, 2]. All implementations (including exhaustive search) cached and reused the pairwise correlation computations where applicable, which was always beneficial for performance. The reported execution time for CD and CDStream corresponds to the total execution cost including the steps of pre-processing, clustering and calculating pairwise correlations. All reported results correspond to averages after 10 repetitions.

Datasets. We include results for three real-world datasets.[4] Results with other datasets had similar qualitative outcomes (see [18] for more details).

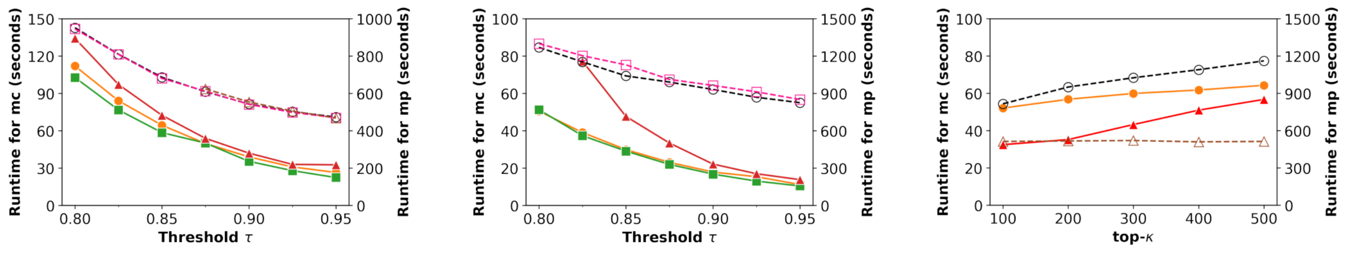
- **Stocks.** Prices of 1596 stocks, covering a period from April 1, 2020 to May 12, 2020. Each stock has its own update frequency, ranging from 1 to 10 minutes. All prices were normalized with log-return normalization, as is standard in finance. To ensure equal dimensionality, all time series were resampled to 5 minute inter-arrival times for CD (leading to 9103 observations), and missing values were filled with standard interpolation. For CDStream, time series were kept at the original update frequency. We used interpolation to fill missing values, which was required for synchronizing the updates. Notice that any algorithm could be used instead for this process, e.g., forward or backward-filling, or even a more complex solution that incorporates ML, e.g., a deep neural network [4].
- **fMRI.** Functional MRI data of a participant watching a movie, prepared with the recommended steps for voxel-based analytics. The data was further pre-processed by mean-pooling with kernels of 2x2x2, 3x3x3, 4x4x4, 6x6x6 and 8x8x8 voxels, each representing the mean activity level at a cube of voxels in the scan. Subsequently, constant-value time series were removed. This led to a total of 9700, 3152, 1440, 509 and 237 time series respectively, all of equal length (5470 observations), covering a period of ~1.5 hours. Unless otherwise mentioned, the reported results correspond to the 4x4x4 resolution, i.e. 1440 time series.
- **SLP.** Sea Level Pressure data [24], as preprocessed in [2]. The dataset contains 171 time series, each with 108 observations.

5.1 CD on static data

5.1.1 Threshold queries. Figs. 4a-b show the effect of threshold τ on execution time of CD for the fMRI and Stocks dataset respectively. The left Y axis corresponds to query $\mathbf{mc}(2, 2)$ with different constraints, whereas the right Y axis corresponds to $\mathbf{mp}(4)$. The plot does not include a result for **mc** in the Stocks dataset for $\tau = 0.8$, since the query returned more than 10 Million results, and our implementation automatically switches to the top-κ variant (with $\kappa = 10^7$) in such cases. Our first observation is that increasing the threshold consistently leads to higher efficiency. This is expected, since a higher threshold enables more aggressive pruning of candidate comparisons. Furthermore, CD is noticeably faster for **mc** compared to **mp**. This is due to two reasons: (a) the complexity of the computation of eigenvalues of a matrix (cubic to $l_{\max}$), which is required for computing the bounds for **mp** (Theorem 3.3), and (b) **mp** typically results in higher correlation values and to more answers for the same value of τ compared to **mc**.

We found that the individual execution times over the 10 repetitions for each configuration were stable, with a relative standard deviation typically between 1%-2%, or below 5 seconds in absolute value. The maximum relative standard deviation for a configuration observed in all experiments was 4.7% of the mean query time.

[4]See https://github.com/CorrelationDetective/public for download links, instructions, and code for reading the data.



(a) mc(2, 2) and mp(4) threshold queries on fMRI **(b) mc(2, 2) and mp(4) threshold queries on Stocks** **(c) mc(2, 2) and mp(4) top-κ queries on fMRI**

Figure 4: Effect of constraints, τ and κ on performance.

5.1.2 Top-κ queries. Fig. 4c shows the execution time of CD for different values of κ with the fMRI dataset – the results with Stocks were qualitatively very similar. We see that a decrease of κ typically leads to increased efficiency. A low value of κ helps the algorithm to increase the running threshold τ faster, leading to more aggressive pruning when Alg. 1 is invoked. Interestingly, this behavior is not as prevalent for **mp**(4) with no constraints. This discrepancy can be attributed to the correlation values in the result set. Indicatively, for this query the lowest correlation in the result set only decreases from 0.917 (top-100) to 0.915 (top-500). In contrast, the same pattern with a minimum jump constraint of $\delta = 0.1$ shows a decrease in this correlation from 0.82 (top-100) to 0.78 (top-500), explaining why the effect of κ on performance is more substantial.

5.1.3 Correlation pattern. Table 2 presents the results size and 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 $O\left(\binom{n}{l_{\max}+r_{\max}}\right)$, 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 **mc**(1, 2) and **mc**(1, 4). Execution time increases by only three orders of magnitude, indicating efficient pruning of the search space.

5.1.4 Clustering sensitivity. We now analyze the sensitivity of CD with respect to the hierarchical clustering parameters. Since correctness of CD is not influenced by the clustering, our experiments only investigate its influence on the efficiency of CD. Table 3 illustrates the effect of K (the number of sub-clusters per cluster) on CD's execution time. A very small number of sub-clusters in each split ($K = 2$) hurts efficiency significantly, as it results in extremely large clusters at the high levels of the hierarchy, and Algorithm 1 needs to drill deeper into the hierarchy before reaching to decisive combinations. Very high K values also lead to suboptimal performance. In that case, the clusters are more compact, leading to decisive combinations at higher levels, but more cluster combinations exist (in these higher levels) that need to be considered.

For K around 10, CD's efficiency is reasonably robust. In fact, setting $K = 10$ led to performance close to the optimal in all cases – at most 15% worse than the optimal performance for the same query, or at most 5% if we do not consider absolute differences up to 10 seconds. The small impact of K, as long as it is not close to the extremes, can be explained by considering how it affects the depth of the clustering tree: intuitively, under the simplifying assumption that each cluster contains approximately an equal amount of vectors, the depth of the clustering hierarchy is approximately $\log_K(n)$. This depth does not vary significantly with the value of K. Indicatively, for 1000 vectors, setting $K \in [10, 30]$ leads to a hierarchy of 3 to 4 levels. Therefore, as long as we avoid extremely small and extremely large K values, the impact of K to CD's efficiency is small. For consistency, for all our remaining experiments we set $K = 10$.

5.1.5 Progressive variant. We also evaluated the progressive nature of CD. We modified our code such that it tracks the number of discovered results at different time points. Figure 5b plots the number of results returned by the algorithm on the Stocks dataset, as a function of time. The results correspond to correlation patterns **mc**(1, 4) and **mp**(4), which take significant time to complete, since these are the ones that would mostly benefit from a progressive algorithm. We see that CD retrieves around half of the results in the first few seconds, and already reaches 80% recall in around 10% of the total execution time.

5.1.6 Comparison to exhaustive search baseline. Figure 5a plots the execution time of CD and the exhaustive baseline for processing fMRI datasets of different sizes, obtained as discussed in Section 5. The results correspond to correlation patterns **mc**(1, 3) and **mp**(3). We see that execution time of CD for both patterns grows at a slower rate compared to the exhaustive search method, and the difference increases with the dataset size. This finding is consistent with our earlier observation that the runtime of CD grows slower than the size of the search space (Section 5.1.3), meaning that CD can handle significantly larger datasets than the exhaustive algorithm in reasonable time.

5.1.7 Comparison to CoMEtExtended. Our next experiment focused on comparing CD with CoMEtExtended [2]. The goal of CoMEtExtended differs slightly from our problem statement. First, CoMEtExtended is approximate. Even though it does not offer approximation guarantees, its recall (and efficiency) 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; when CoMEtExtended is configured to return the exact result set ($\rho_{\text{CE}} = 1$), it degenerates to exhaustive search [2], to which we compared in Section 5.1.6. In contrast, CD always produces complete answers. Therefore, we consider both execution time and recall rate in our comparison. Second, CoMEtExtended aims to find only *maximal* sets that exhibit a strong **mp** correlation, whereas CD finds *all* sets (up to a specified cardinality) that are strongly correlated. To

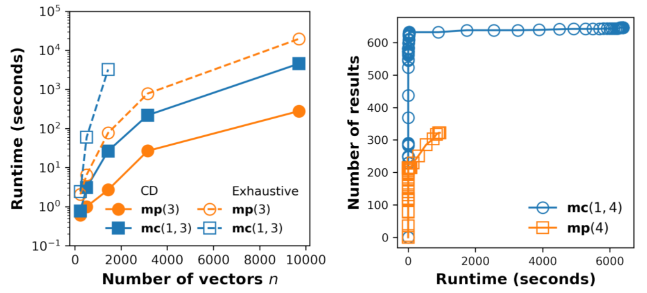


Figure 5: (a) Running time of CD (filled markers) and exhaustive algorithm (empty markers, dashed lines) for varying resolutions of the fMRI dataset, with $\tau = 0.9$ and no constraints. Queries were interrupted after 20 hours. (b) Number of retrieved results in relation to runtime, for progressive execution of $\mathbf{mc}(1, 4)$ and $\mathbf{mp}(4)$, with $\tau = 0.9, \delta = 0.05$ on the Stocks dataset.

ensure a fair comparison for CoMEtExtended, we also considered all subsets of each result 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 step was not included in the execution time of CoMETExtended, i.e., it did not penalize its performance. Conversely, instead of enhancing the results of CoMETExtended we could filter out the non-maximal results from CD's result set. Since both approaches led to a very similar comparison (recall and execution time), we present only the results of the first approach. Table 5 presents the number of results and execution time of CoMEtExtended and CD on the same dataset (SLP) and parameters used in [2]. We only consider the **mp** measure, since CoMEtExtended does not support **mc**. We see that CD is consistently faster than CoMEtExtended – at least an order of magnitude – and often returns substantially more results. Indicatively, for $\mathbf{mp}(4)$, 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 $\delta = 0.1$, CoMEtExtended found 281 results with 6 vectors, and one with 7 ($\rho_{CE} = 0.02, \tau = 0.4$). These amount to $\sim 0.3\%$ of the total amount of discovered results. These were not discovered by CD, as the queries specified $l_{\max} = 5$ at most, prioritizing the simpler and more interpretable results. Nevertheless, for these settings, CD still found 25% more results than COMEtExtended, and in one fourth of the time. Moreover, the case studies presented in [1, 2], 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 [5]. For these cases, with a fixed $l_{\max}$, CD is guaranteed to find a superset of COMEtExtended's result set, at a fraction of the time.

5.1.8 Comparison to CONTRa. We also compared CD to CONTRa [1] for discovery of tripoles, i.e., $\mathbf{mc}(1, 2)$ correlations. For 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 only one thread/core, since the implementation of CONTRa was single-threaded. CONTRa was configured to return the exact results.

The experimental results with the fMRI dataset are shown in Table 6.[5] We see that CD is more efficient than CONTRa for detecting the same results, even with $\tau = 0$. However, the lack of τ 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 [1]) and $\tau = 0.9$ (which gives a reasonable amount of results, in the order of a few tens to hundreds). This further decreases the runtime of CD by one to two orders of magnitude, while preventing clutter of the result set by returning only the most strongly correlated triplets.

5.2 Evaluation with streaming data

The second set of experiments was configured to evaluate the performance of CDStream. We used the timestamps that are contained in the three datasets for ordering the data and generating the streams. Our discussion will focus on the Stocks dataset; results for the other datasets are shown only when they offer new insights. Unless otherwise mentioned, the following results correspond to a batch size of 50, a sliding window of 2000, and a dataset size of 1000 stocks.

5.2.1 Comparison to CD. Fig. 6a presents CDStream's mean processing time per batch, for different dataset sizes created by randomly picked stocks. The figure also includes the average time required for executing CD at the end of each batch. We see that CDStream is more efficient than CD for small correlation patterns, requiring a few milliseconds. Note that, even though the number of comparisons increases at a combinatorial rate with the number of vectors , the execution time of CDStream grows substantially slower. This is due to the grouping technique in the index of CDStream, which effectively reduces the work for processing each update. For more complex patterns, e.g., $\mathbf{mc}(2, 3)$, CDStream has performance comparable to CD.

5.2.2 Effect of query parameters. Table 4 presents the effect of τ and constraints (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 combinations 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.

Figure 6b plots the average processing time per update, for varying batch sizes and for both fMRI and Stocks. The batch size (X-axis) is presented as a multiplicative factor on the number of vectors n in each dataset. We see that the batch size enables tuning the tradeoff between throughput and update rate of the results: increasing the batch size increases efficiency, but reduces freshness of results. This happens because both algorithms will process only the latest values for each vector, ignoring intermediary updates. Also observe that CD's efficiency approaches that of CDStream as the batch size increases. For Stocks, processing time for the two algorithms crosses at a batch size $4 * n$, whereas for fMRI, this crossing happens at batch size $8 * n$. This discrepancy can be attributed to the properties of the datasets (the inherent distributions and magnitude of updates) and exemplifies the importance of CDHybrid. As we will see shortly (Section 5.2.4), CDHybrid is able to adapt to the

[5]For this experiment, the minimum jump parameter δ is defined as in [1], to represent the minimum difference between the *squared* correlations.

Table 2: CD with different correlation patterns.

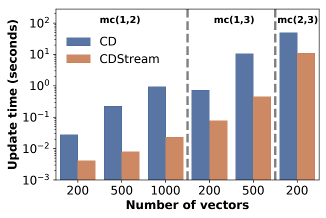
	fMRI		Stocks	
	time (s)	#results	time (s)	#results
$\mathbf{mc}(1,2)$	1.4	53	2.0	581
$\mathbf{mc}(1,3)$	26.8	1350	23.3	632
$\mathbf{mc}(2,2)$	41.5	4239	18.2	1875
$\mathbf{mc}(1,4)$	6294	42196	6369.9	646
$\mathbf{mc}(2,3)$	15760	287651	4238.6	2796
$\mathbf{mp}(3)$	2.6	33	4.6	302
$\mathbf{mp}(4)$	560.0	58213	966.4	576

Table 3: Execution times (in seconds) for varying clustering parameters and queries ($\delta = 0.05$).

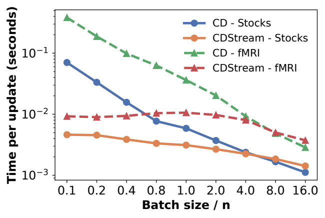
		fMRI					Stocks				
	$\tau \backslash K$	2	5	10	25	50	2	5	10	25	50
$\mathbf{mc}(1,3)$	0.8	446	**108**	121	131	157	722	106	106	142	**104**
	0.9	140	**35**	40	49	63	281	**23**	27	48	36
$\mathbf{mc}(2,2)$	0.8	883	**174**	188	177	197	715	78	**75**	92	80
	0.9	264	**57**	64	68	94	179	**22**	**22**	35	30
$\mathbf{mp}(4)$	0.8	4799	1037	**1014**	1061	1149	10547	**1366**	1369	1809	1424
	0.9	2451	**592**	606	641	706	6808	1020	**981**	1497	1200

Table 4: Effect of τ and δ on CD and CDStream for streaming data, with Stocks.

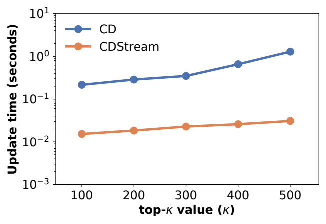
	CD			CDStream		
$\delta \backslash \tau$	**0.6**	**0.7**	**0.8**	**0.6**	**0.7**	**0.8**
None	3.12	2.62	0.80	.045	.036	.023
Irred.	3.21	3.26	0.88	.046	.036	.023
0.05	3.87	2.39	0.93	.043	.034	.023
0.10	3.00	2.77	1.13	.044	.033	.023
0.15	3.15	2.49	1.14	.043	.033	.022



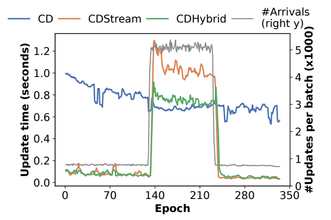
(a) Effect of dataset size and correlation pattern, with $\delta = 0.05$, $\tau = 0.8$, Stocks.



(b) Effect of batch size, $\delta = 0.05$, $\tau = 0.8$.



(c) Effect of κ, with $\delta = 0.05$, with Stocks.



(d) Efficiency of CDHybrid over time, with Stocks.

Figure 6: Effect of query parameters on performance of CDStream.

Table 5: Comparison of CoMEtExtended and Correlation Detective on SLP: running time (seconds) and number of retrieved results.

	CoMEtExtended						Correlation Detective			
τ, δ	$\rho_{\text{CE}} = 0$		$\rho_{\text{CE}} = 0.01$		$\rho_{\text{CE}} = 0.02$		$\mathbf{mp}(4)$		$\mathbf{mp}(5)$	
	time	#res.	time	#res.	time	#res.	time	#res.	time	#res.
0.4, 0.1	604	62663	1318	67110	3530	70921	11	71083	899	88305
0.4, 0.15	511	7244	1218	7300	3393	7343	9	7559	575	7562
0.4, 0.2	501	2166	1210	2171	3327	2174	7	2183	333	2183
0.5, 0.1	459	30632	1099	33718	2836	36457	7	34592	557	51391
0.5, 0.15	398	3646	1006	3702	2760	3745	6	3961	391	3964
0.5, 0.2	390	1434	1006	1439	2701	1442	6	1451	292	1451
0.6, 0.1	246	7823	598	8892	1592	9859	5	9204	310	17349
0.6, 0.15	223	1569	577	1606	1559	1635	5	1840	245	1843
0.6, 0.2	219	771	568	776	1532	779	5	788	199	788

Table 6: Comparison of CONTRa and CD: running time (seconds) and number of results for the largest fMRI dataset ($n = 9700$).

	CONTRa		CD ($\tau = 0$)		CD ($\tau = 0.5$)		CD ($\tau = 0.9$)	
δ	time	results	time	results	time	results	time	results
0.1	>24hrs	22952036	17027	22952036	3602	20527560	458	432
0.15	11162	733018	7168	733018	3151	732908	458	102
0.2	5324	20555	3852	20555	2790	20555	459	24

properties of the dataset, and chooses the best algorithm. In [18] we also report on the sensitivity of CDStream to the sliding window size, and consider more values for batch size.

5.2.3 Top-κ queries. Fig. 6c plots the average processing time per batch for top-κ query $\mathbf{mc}(1,2)$, for different κ values. We see that processing time for both algorithms increases with κ. In CD, execution time grows almost linearly with κ (from 200 msec to almost 1.3 second), whereas for CDStream the time increases by only a factor of two for the same values. 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 to avoid finding less than κ results.

5.2.4 CDHybrid. For this experiment, we use a time-based batch size of 1 minute, and simulate stream bursts by speeding up the updates (reducing the inter-arrival times) around the middle of the stream, for approximately one third of the stream length. Figure 6d depicts the processing time per batch (moving average for the last 5 batches), for processing Stocks with CD, CDStream, and CDHybrid. Each epoch corresponds to one batch. The figure also includes the number of arrivals within each batch (right Y axis). We observe that CDHybrid quickly switches to the best method. Shortly after a switch from CDStream to CD, the cost of CDHybrid is slightly higher compared to the optimal cost. This is attributed to the initialization cost of CD. For the case of switching back to CDStream (epoch 240), the additional cost for updating the outdated index is also small, indicating that the process of updating the index after the switch is not expensive. Also recall that part of this cost (for removing the expired decisive combinations from the index) is amortized through a large number of epochs, due to the lazy updating algorithm discussed in Section 4.2. Particularly, switching to CDStream has a cumulative cost of 0.109 seconds, amortized over 60 epochs, amounting to $\sim 3.8\%$ of the total processing time over these epochs. The cost of CDHybrid to decide between the two algorithms was negligible in all cases, requiring less than 0.1 msec. This cost is already included in the shown results.

6 CONCLUSIONS

We considered the problem of detecting high multivariate correlations with two 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-world datasets showed that our contribution outperforms the state of the art typically by an order of magnitude.

REFERENCES

[1] Saurabh Agrawal, Gowtham Atluri, Anuj Karpatne, William Haltom, Stefan Liess, Snigdhansu Chatterjee, and Vipin Kumar. 2017. Tripoles: A New Class of Relationships in Time Series Data. In *Proceedings of the 23rd SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 697–706.

[2] Saurabh Agrawal, Michael Steinbach, Daniel Boley, Snigdhansu Chatterjee, Gowtham Atluri, Anh The Dang, Stefan Liess, and Vipin Kumar. 2020. Mining Novel Multivariate Relationships in Time Series Data Using Correlation Networks. *IEEE TKDE* 32, 9 (2020), 1798–1811.

[3] David Arthur and Sergei Vassilvitskii. 2007. K-Means++: the advantages of careful seeding. In *Proc. 18th Annual Symposium on Discrete Algorithms, SODA*, Nikhil Bansal, Kirk Pruhs, and Clifford Stein (Eds.). SIAM, 1027–1035.

[4] Wei Cao, Dong Wang, Jian Li, Hao Zhou Bytedance, A I Lab, Yitan Li, Bytedance Ai Lab, and Lei Li. 2018. BRITS: Bidirectional Recurrent Imputation for Time Series. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Montréal, Canada). 6776–6786.

[5] Roger H.L. Chiang, Chua Eng Huang Cecil, and Ee-Peng Lim. 2005. Linear correlation discovery in databases: a data mining approach. *Data & Knowledge Engineering* 53, 3 (2005), 311–337.

[6] Abhimanyu Das and David Kempe. 2008. Algorithms for Subset Selection in Linear Regression. In *Proc. 40th ACM Symposium on Theory of Computing (STOC '08)*. ACM, 45–54.

[7] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-Sensitive Hashing Scheme Based on p-Stable Distributions. In *Proc. 20th Annual Symposium on Computational Geometry (SCG '04)*. ACM, 253–262.

[8] Ronald Fagin, Amnon Lotem, and Moni Naor. 2001. Optimal Aggregation Algorithms for Middleware. *J. Comput. System Sci.* 66 (2001), 614–656.

[9] Simons Foundation. 2021. SPARK for Autism. https://sparkforautism.org/portal/page/autism-research/. Accessed: 2021-07-30.

[10] Simons Foundation. 2021. SPARK Gene list. https://d2dxtcm9g2oro2.cloudfront.net/wp-content/uploads/2020/07/13153839/SPARK_gene_list_July2020.pdf. Accessed: 2021-07-30.

[11] Daniel A. Handwerker, Vinai Roopchansingh, Javier Gonzalez-Castillo, and Peter A. Bandettini. 2012. Periodic changes in fMRI connectivity. *NeuroImage* 63, 3 (2012), 1712–1719.

[12] Stephan Heunis, Rolf Lamerichs, Svitlana Zinger, Cesar Caballero-Gaudes, Jacobus F.A. Jansen, Bert Aldenkamp, and Marcel Breeuwer. 2020. Quality and denoising in real-time functional magnetic resonance imaging neurofeedback: A methods review. *Human Brain Mapping* 41, 12 (2020), 3439–3467.

[13] Wolfgang Karl Härdle. 2007. *Applied Multivariate Statistical Analysis* (2 ed.). Springer. 321–330 pages.

[14] Silvan Licher, Shahzad Ahmad, Hata Karamujić-Čomić, Trudy Voortman, Maarten J. G. Leening, M. Arfan Ikram, and M. Kamran Ikram. 2019. Genetic predisposition, modifiable-risk-factor profile and long-term dementia risk in the general population. *Nature Medicine* 25, 9 (2019), 1364–1369.

[15] Stefan Liess, Saurabh Agrawal, Snigdhansu Chatterjee, and Vipin Kumar. 2017. A Teleconnection between the West Siberian Plain and the ENSO Region. *Journal of Climate* 30, 1 (2017), 301 – 315.

[16] Myles E. Mangram. 2013. A Simplified Perspective of the Markowitz Portfolio Theory. *Global Journal of Business Research* 7, 1 (2013), 59–70.

[17] Fukuda Megumi, Ayumu Yamashita, Mitsuo Kawato, and Hiroshi Imamizu. 2015. Functional MRI neurofeedback training on connectivity between two regions induces long-lasting changes in intrinsic functional network. *Frontiers in Human Neuroscience* 9 (2015).

[18] Koen Minartz, Jens d'Hondt, and Odysseas Papapetrou. 2021. *Multivariate correlation discovery in static and streaming data*. Technical Report. Eindhoven University of Technology. Available in https://github.com/CorrelationDetective/public.

[19] Ileena Mitra, Alinoë Lavillaureix, Erika Yeh, Michela Traglia, Kathryn Tsang, Carrie E. Bearden, Katherine A. Rauen, and Lauren A. Weiss. 2017. Reverse Pathway Genetic Approach Identifies Epistasis in Autism Spectrum Disorders. *PLOS Genetics* 13, 1 (01 2017), 1–27. https://doi.org/10.1371/journal.pgen.1006516

[20] Abdullah Mueen, Suman Nath, and Jie Liu. 2010. Fast Approximate Correlation for Massive Time-Series Data. In *Proc. ACM International Conference on Management of Data (SIGMOD '10)*. ACM, 171–182.

[21] Hoang Vu Nguyen, Emmanuel Müller, Periklis Andritsos, and Klemens Böhm. 2014. Detecting Correlated Columns in Relational Databases with Mixed Data Types. In *Proc. 26th International Conference on Scientific and Statistical Database Management (SSDBM '14)*. ACM, Article 30, 12 pages.

[22] Hoang Vu Nguyen, Emmanuel Müller, Jilles Vreeken, Pavel Efros, and Klemens Böhm. 2014. Multivariate Maximal Correlation Analysis. In *Proc. 31st International Conference on Machine Learning - Volume 32 (ICML'14)*. 775–783.

[23] Örjan Carlborg and Chris S. Haley. 2004. Epistasis: too often neglected in complex trait studies? *Nature Reviews Genetics* 5, 8 (2004), 618–625.

[24] Kistler RE, Eugenia Kalnay, William Collins, Suranjana Saha, G. White, John Woollen, Muthuvel Chelliah, Wesley Ebisuzaki, Masao Kanamitsu, Vernon Kousky, Huug Dool, Jenne RL, and Mike Fiorino. 2001. The NCEP/NCAR 50-year reanalysis: monthly means CD-ROM and documentation. *Bulletin of the American Meteorological Society* 82 (2001), 247–268.

[25] Camilo Rostoker, Alan Wagner, and Holger Hoos. 2007. A Parallel Workflow for Real-time Correlation and Clustering of High-Frequency Stock Market Data. In *Proc. 21th International Parallel and Distributed Processing Symposium*. 1–10.

[26] Venu Satuluri and Srinivasan Parthasarathy. 2012. Bayesian Locality Sensitive Hashing for Fast Similarity Search. *Proc. VLDB Endow.* 5, 5 (2012), 430–441.

[27] Zhiyuan Tan, Aruna Jamdagni, Xiangjian He, Priyadarsi Nanda, and Ren Ping Liu. 2014. A system for denial-of-service attack detection based on multivariate correlation analysis. *Trans. Parallel and Distributed Systems (TPDS)* 25, 2 (2014), 447–456.

[28] Satosi Watanabe. 1960. Information Theoretical Analysis of Multivariate Correlation. *IBM Journal of Research and Development* 4, 1 (1960), 66–82.

[29] Yingjun Wu, Jia Yu, Yuanyuan Tian, Richard Sidle, and Ronald Barber. 2019. Designing Succinct Secondary Indexing Mechanism by Exploiting Column Correlations. In *Proc. International Conference on Management of Data (SIGMOD'19)*. ACM, 1223–1240.

[30] Xiang Zhang, Feng Pan, Wei Wang, and Andrew Nobel. 2008. Mining non-redundant high order correlations in binary data. *Proc. VLDB Endow.* 1, 1 (2008), 1178–1188.

[31] Yunyue Zhu and Dennis Shasha. 2002. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. In *Proc. 28th International Conference on Very Large Data Bases (VLDB '02)*. 358–369.

[32] Anna Zilverstand, Bettina Sorger, Jan Zimmermann, Amanda Kaas, and Rainer Goebel. 2014. Windowed Correlation: A Suitable Tool for Providing Dynamic fMRI-Based Functional Connectivity Neurofeedback on Task Difficulty. *PLOS ONE* 9, 1 (01 2014), 1–13.