Multivariate correlations discovery in static and streaming data

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

Jens d'Hondt
Eindhoven University of Technology
j.e.dhondt@student.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 the data and extracting salient insights. Most works to date focus on detecting high pairwise correlations – pairs of variables/vectors that have a high Pearson correlation coefficient. A generalization of this problem with known applications but no known efficient solutions involves discovery of significant multivariate correlations, i.e., select 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 rely on novel theoretical results, work with two different correlation measures, and support additional constraints. Our extensive experimental evaluation with 5 datasets 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 d'Hondt, and Odysseas Papapetrou. Multivariate correlations discovery in static and streaming data. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

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

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 [10]. In finance, correlation plays a crucial role in finding portfolios of assets that are on the Pareto-optimal frontier of risk and expected returns [17], and in genetics, correlations help scientists detect cause factors for potentially hereditary syndromes.[1] Correlations – as a generalization of functional dependencies – also found use for optimizing access paths in databases [31].

Multivariate, or high-order correlations, are a generalization of pairwise correlations that can capture relations of arbitrarily-sized sets of variables, represented either as high-dimensional vectors or as time series.[2]

In the last few years, multivariate correlations found extensive use in diverse domains. Detection of ternary correlations in fMRI time series improved our understanding of how different brain regions work in cohort for executing different tasks [1, 2]. In climatology a ternary correlation led to the characterization of a new weather phenomenon and to improved climate models [16]. It is also stipulated that a more thorough look at multivariate correlations will open doors in the fields of genomics [25, 32] and medicine [15, 19].

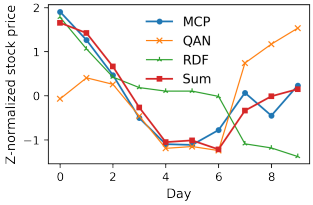
Accordingly, several measures and algorithms for discovering strong multivariate correlations have been proposed. Agrawal et al. [1] defined tripoles – sets of three vectors – where the sum of the first two vectors has a high linear correlation with the third one. More recently, the multipoles measure was introduced, which measures the degree of linear dependence of a vector set of arbitrary cardinality [2]. Canonical Correlation Analysis (CCA) operates on two sets of vectors, and finds linear combinations of each set such that the Pearson correlation of the two resulting vectors is maximized [13]. Non-linear multivariate correlation measures are also frequently used. For example, Total Correlation is an information theoretic measure that expresses the amount of information shared between variables [30], which serves as the basis for several correlation measures and algorithms [22, 23, 32].

The fundamental impediment on the discovery of strong multivariate correlations in datasets with many vectors is the vast search space – the possible combinations of vectors that need to be examined. Unfortunately, apriori-like pruning techniques do not apply for the general case of multivariate correlations. Consider, for example, the three time series presented in Figure 1, which represent closing prices of three stocks from the Australian securities exchange. In this example, the pairwise correlation between all pairs of the three time series is 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. At the same time, 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. Smart algorithms are needed that can drastically prune the search space to reduce computational complexity.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

[1] A prime example is the Spark project for discovering gene properties related to the manifestation of the autism spectrum disorder [8], which led to a list of genes and their correlated symptoms [9]

[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.



	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.

Past algorithms follow at least one of the following approaches: (a) they focus on a definition of multivariate correlations that enables an apriori-like filtering [2, 22, 32], (b) they rely on hand-crafted additional assumptions of the user query, which may be too constraining for other application scenarios [1, 2, 32], or, (c) they do not guarantee completeness of the result set [1, 2]. Consequently, even though these algorithms are still very useful for their particular use cases, they are not relevant for general use.

In this work, we follow a more general direction. First, we also consider correlation measures that are, by nature, not suitable for apriori-like pruning. Our algorithm prunes the search space by identifying clusters of vectors, and handling all vectors within the cluster as a single entity. Moreover, cluster comparisons are executed in such a way that their complexity does not depend on the dimensionality d of each vector. 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.

We consider different algorithmic variants: an exact threshold variant that returns all correlations higher than a threshold τ, and an exact top-k variant that returns the top-k highest correlations. We also discuss the case of progressively finding results. Finally, we extend the proposed algorithms to a dynamic context, enabling efficient handling of streaming data. This opens the door to use-cases where continuous updates of query answers are required, such as flash-trading models in finance [27], weather- and server monitoring [29] and neurofeedback training [12, 18, 34].

We evaluate our algorithms on 5 datasets, and compare them to the state-of-the-art. Our evaluation demonstrates that we outperform the state-of-the-art 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 we will be considering in this work. We then formalize the problem, and discuss prior work and other multivariate correlation measures.

2.1 Correlation Measures

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

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

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

where ρ denotes the Pearson correlation coefficient and $\hat{\mathbf{x}}$ denotes $\mathbf{x}$ after z-normalization, i.e., $\hat{\mathbf{x}}_i = \frac{\mathbf{x}_i - \mu_{\mathbf{x}}}{\sigma_{\mathbf{x}}}$. Intuitively, **mc** takes the element-wise aggregates of the z-normalized vectors in X and those in Y, and calculates the Pearson correlation of the result. The default aggregation method is average (as in Eqn. 1), but both the definition and our work can be easily extended to weighted linear aggregates. Tripoles [1] is a special case of the multiple correlation measure, where $|X| = 2$ and $|Y| = 1$. In this work, we adopt the unconstrained definition, allowing both X and Y to contain more vectors.

Multipoles. Multipoles correlation $\mathbf{mp}(\cdot)$ measures the linear dependence of an input set of vectors [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:

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

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 $\mathbf{X}\cdot\mathbf{v}$ has zero variance.

Notice that multipoles and multiple correlation are not equivalent, and the one is not a generalization of the other. By definition, multipoles assumes optimal weights (vector $\mathbf{v}$ is such that the variance is minimized), whereas for the case of multiple correlation, the aggregation function for the vectors (e.g., averaging) is determined at the definition of the measure and is independent of the vectors. Furthermore, multipoles expresses the degree of linear dependence within a single set of vectors, whereas for multiple correlation, 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 $Corr$ defined over this space, 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, to avoid discussing the two cases separately, we will be denoting the correlation function with $Corr(X, Y)$, 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-k query For a user-chosen correlation function $Corr$, integer parameter k, and parameters $l_{\max}, r_{\max} \in \mathbb{N}$, find the k 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 understand, and more useful to the data analyst. Complementary to the two query types, users may also want to specify a

set of additional constraints. Typically, these constraints relate to the targeted diversity of the answers. We will consider two different constraints, but other constraints (e.g., the weak-correlated feature subset constraint of [32]) 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 smaller answers, which are more easily explainable.

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], prioritizes the solutions where each vector in X and Y contributes at least δ to the increase of the correlation.

The minimum jump constraint applies to both query types. The irreducibility constraint on the other hand is useful only for threshold queries. With top-k queries, irreducibility is ill-defined. For example, 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-k 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 follows: $\mathbf{mc}(l_{\max}, r_{\max})$ (for **mc**) and $\mathbf{mp}(l_{\max})$ for (for **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 **mc** correlation. Pattern $\mathbf{mp}(4)$ will identify 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

Correlation analysis is commonly used in Exploratory Data Analysis, and several algorithms exist for finding highly correlated pairs in large data sets of high-dimensional vectors, e.g., time series. Zhu and Sasha [33] developed StatStream, a tool for discovering highly correlated pairs in streaming time series data. The core ideas of StatStream are: (a) a monotonic one-to-one mapping of Pearson correlation to Euclidean distance, and, (b) Discrete Fourier Transform (DFT) to reduce the dimensionality of the vectors, such that these can be indexed in a low-dimensional grid, where all highly-correlated pairs end up in neighboring cells. Mueen et al. [21] also map pairwise correlations to Euclidean distance and exploit DFT, but instead resort to dynamic programming to reduce the number of comparisons. Finally, several works enable indexing of high-dimensional vectors in the Euclidean space [6, 28]. In combination to the one-to-one mapping between Pearson correlation and Euclidean distance, these indices can be exploited to find highly-correlated pairs. However, since these works focus on pairwise correlations, they rely on the premise that in a dataset of n vectors, an index needs to contain only $O(n)$ vectors. For the case of multivariate correlations, the number of possible combinations increases combinatorially with the values of $l_{\max}$ and $r_{\max}$. Therefore, enumerating these vector combinations and indexing them becomes prohibitively expensive. Also, an index on the individual vectors is not helpful 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).

Prior works addressing multivariate correlations propose algorithms that rely on additional constraints for their pruning power. For the tripoles metric, Agrawal et al. investigate the problem of finding highly-correlated tripoles [1]. A tripole is a special case of the multiple correlation measure that contains only three vectors. The pruning power of their algorithm relies on the minimum jump constraint. Compared to tripoles, our work handles the more general definition of the multiple correlation measure, allowing aggregation of more vectors at the left and right hand side. Furthermore, our work does not rely solely on the minimum jump constraint to prune comparisons. Instead, it relies on novel theoretical results to prune large groups of combinations without actually enumerating over each individual combination.

Algorithms for discovering high correlations according to the multipole measure (Eqn. 2) were first proposed in [2]. Both CoMEt and CoMEtExtended are approximate algorithms that use clique enumeration to efficiently explore the search space of all possible combinations. Their efficiency depends on a parameter ρ that trades off completeness of the result set 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 significantly more complete result set compared to baseline methods like l_1-regularization based techniques, and methods from the field of structure learning. 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.

Total correlation is a non-linear information-theoretic metric that expresses how much information is shared between variables [13]. Nguyen et al. [22] 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, and that consequently yield a high total correlation value. Although the method discovers interesting sets of vectors, as seen in the example of Fig. 1, 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. developed an algorithm that discovers sets with a high total correlation value [32]. The algorithm bounds correlations using two theoretical results: one is based on the pairwise correlations within the set, while the other bounds the correlation difference of sets that are highly similar. However, the method is limited to data with binary features, like gene expression data. Furthermore, it relies on the additional *weak-correlated subset constraint*, which dictates that each of the answer's subsets must have a correlation of at most a user-defined threshold α. Depending on the context, such a constraint may not be desired.

In the supervised learning context, subset regression is related to multivariate correlation mining. The goal of this feature selection problem is to select the best p predictors out of n candidate

features [5]. Heuristics like forward selection, backward elimination or genetic algorithms are commonly used to find good solutions [11, 20]. Our problem differs from the above in two aspects. First, we aim to find interesting patterns in the data, instead of finding the best predictors for a given dependent variable. Second, instead of finding only the single highest correlated set of vectors, our goal is to find a *diverse set* of results. Showing multiple diverse results to the domain expert helps her to further assess the results on qualitative aspects and to gain more insights.

3 DETECTION OF MULTIVARIATE CORRELATIONS ON STATIC DATA

The main challenge in detecting strongly correlated vector sets stems from the combinatorial explosion of the number of possible combinations that need to be examined. In a dataset of n vectors, there exist at least $O\left(\sum_{i=2}^{p}\binom{n}{k}\right)$ possible combinations, where $p = l_{\max} + r_{\max}$ denotes the maximum number of vectors that can participate in a correlation. This means that, even if each possible combination can be checked in constant time, the enumeration of all combinations will still take significant time.

Our algorithm – Correlation Detective, abbreviated as *CD* – exploits the insight that vectors often exhibit (possibly weak) correlations between them. For example, securities of companies that participate in 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 an increased correlation between their stock prices. Still, current algorithms always consider the input vectors independently when examining all possible combinations. Our algorithm exploits these correlations, even if they are weak, to drastically reduce the search space.

CD works as follows: rather than enumerating over all possible vector combinations that correspond to the correlation pattern, CD clusters vectors based on their similarity, and enumerates over the combinations of only the cluster centroids, which are generally several orders of magnitude less. 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 already be added to the result set, can safely be discarded, or, finally, if the clusters should be split further into smaller subclusters, for deriving tighter bounds. This effectively reduces the number of combinations that need to be considered, making CD an order of magnitude faster than the competitive algorithms.

In the remainder of this section, we will present the elements of CD, explaining how the two types of queries presented in Section 2 are handled. We will start with a brief description of the initialization phase, which includes data pre-processing and the clustering algorithm. In Sections 3.1 and 3.2 we will describe how CD answers threshold and top-k queries respectively.

Initialization phase. As a first step, all vectors are z-normalized, i.e., shifted and scaled such that they have a zero mean and a standard deviation equal to 1. After this phase the algorithm operates only on z-normalized vectors. We will simply denote these as $\mathbf{x}$ instead of $\hat{\mathbf{x}}$. The next step involves hierarchical clustering of the data. We start with a selection of c initial centroids, chosen with K-Means++ [3]. Each vector is then assigned to the closest cluster centroid, with distance computed in the Euclidean space. Each cluster is recursively broken to c sub-clusters using the same algorithm, until we reach clusters with a single vector. Notice that, unlike the typical K-Means clustering, our implemented algorithm does not repeat until convergence. Our experiments with different datasets have shown that a single iteration is sufficient for decent clustering. Running more iterations does not lead to a noticeable increase of the performance of CD. Clustering requires computation of pairwise distances between vectors, which are also cached, such that they can be reused at later stages of the algorithm. Overall the running time of the whole initialization phase takes a negligible fraction of the total running time of CD.

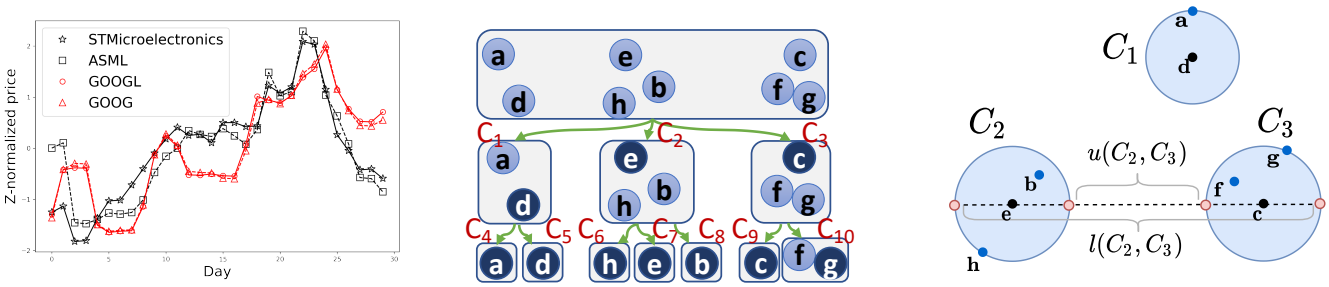
3.1 Threshold queries

CD receives as input the cluster tree produced by the hierarchical clustering algorithm, a correlation pattern (e.g., $\mathbf{mc}(2,1)$, which restricts the correlations to types $(\mathbf{x},\mathbf{y})$ and $((\mathbf{x},\mathbf{y}),\mathbf{z})$, where $\mathbf{x},\mathbf{y},\mathbf{z}$ are placeholders for vectors), a correlation function *Corr*, and a correlation threshold τ. It will then start from the children clusters of the root, forming all possible combinations of the correlation pattern with these. In the example of Fig. 2(b), which we will be using as a running example, if the desired correlation pattern is $\mathbf{mc}(2,1)$, the following *combinations of clusters* will be examined in 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 cluster combination, the algorithm will compute lower and upper bounds on the correlation of these clusters, denoted with LB and UB respectively (Alg. 1, line 1). We will explain shortly how these bounds can be computed efficiently. 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 is *decisive negative* – any materialization cannot yield a correlation higher than the threshold τ. Therefore, this combination does not need to be examined further. 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. 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 (the maximum Euclidean distance from the centroid to any other vector in the cluster), finds its sub-clusters from the clustering tree, and recursively checks all combinations where $C_{\max}$ is replaced by a sub-cluster. In the running example, 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 the three children of C_2, namely C_6, C_7, C_8, and examine their combinations with C_1 and C_3. This process continues recursively until each combination is decisive. Decisive combinations are typically found at high levels of the cluster tree, thereby saving many comparisons.



(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 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 3.1.

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 τ.

```
(LB, UB) ← CalcBounds(S_l, S_r, Corr)
if LB ≥ τ then
    Add (S_l, S_r) to the result set
else if UB < τ then
    Discard (S_l, S_r)
else
    // Replace largest cluster with subclusters and recurse
    C_max ← arg max_{C ∈ S_l ∪ S_r} {C.radius}
    Set SC ← C_max.subclusters
    for S ∈ SC do
        (S'_l, S'_r) ← (S_l, S_r) with C_max replaced by S
        ThresholdQuery((S'_l, S'_r), Corr, τ)
```

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. 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.

3.1.1 Theoretical bounds. We first present a lemma for bounding the Pearson correlation between only two clusters. This will serve as a stepping stone for multivariate correlations. For ease of exposition, the lemma uses trigonometric functions. It can be easily rewritten to use dot products, by observing that the cosine of the angle between two vectors $\mathbf{x}$ and $\mathbf{y}$ is equal to their normalized dot product, i.e., $cos(\theta_{\mathbf{x},\mathbf{y}}) = \frac{\mathbf{x}\cdot\mathbf{y}}{||\mathbf{x}||_2||\mathbf{y}||_2}$.

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

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

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

Proof. All proofs are included in the appendices. □

The above lemma allows us to bound the unknown correlation between two vectors $\mathbf{u_1}$ and $\mathbf{u_2}$ that belong in two clusters with centroids $\mathbf{v_1}$ and $\mathbf{v_2}$ respectively, by using: (a) the angle formed by the two centroids, and, (b) upper bounds for the angles formed by $\mathbf{u_1}$ with its centroid $\mathbf{v_1}$, and by $\mathbf{u_2}$ with its centroid $\mathbf{v_2}$. For instance, in our running example (Fig. 2(b)), we can bound the correlation between **a** and **b** if we have the cosine of the two cluster centroids **d** and **e**, and the cosines of **a** with **d** and **h** with **e** (as **h** is the furthest point in C_2 from the centroid **e**). Furthermore, the bounds will tighten if the maximum angles formed by each centroid with all vectors in the cluster are reduced.

We now extend our discussion to cover multivariate correlations, which involve 3 or more clusters. As correlation measures, we will use the two measures discussed in Section 2: Multiple Correlation **mc** (Theorem 3.2), and Multipoles **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 valid lower/upper bounds on the pairwise correlations between the clusters' contents, 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:*

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

Theorem 3.2 can be used to efficiently bound the multiple correlation of any combination of clusters that satisfies the correlation pattern, without testing all possible materializations of this combination. For example, if we are considering the cluster combination $((C_1, C_2), C_3)$ from our running example, we can use Lemma 3.1 to calculate bounds for pair C_1, C_2, such that $\forall \mathbf{x} \in C_1, \mathbf{y} \in C_2 : l(C_1, C_2) \leq \rho(\mathbf{x}, \mathbf{y}) \leq u(C_1, C_2)$, and similar for pairs C_1, C_3 and C_2, C_3. These pairwise bounds take $O(1)$ to compute per pair of clusters, independent of the size of the clusters.

An interesting observation here is that by tightening the bounds for the pairwise correlations, we can reduce $L(\cdot, \cdot)$ and $U(\cdot, \cdot)$, which will in turn tighten the bounds for **mc**. We will exploit this observation in the next section.

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 valid lower/upper bounds on the pairwise correlations between the cluster's vectors, 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 $\mathcal{S} = \{C_1, C_2, \ldots, C_{l_{max}}\}$. Furthermore, let $\mathbf{L}$ and $\mathbf{U}$ be symmetric matrices such that $l_{ij} = l(C_i, C_j)$ and $u_{ij} = u(C_i, C_j)$ for all $1 \leq i, j \leq n$. 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:*

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

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

Similar to the bounds for **mc**, this theorem requires valid lower and upper bounds for the pairwise correlations between clusters, which can again be derived with Lemma 3.1. The tightness of these pairwise bounds will determine the l_2-norm of $\mathbf{U} - \mathbf{L}$, which in turn determines the tightness of the multipoles bound.

The proofs of both Theorems (included in the technical report) rely on rewriting the multivariate correlations to use the pairwise correlations, which can be bounded using Lemma 3.1.

3.1.2 Empirical pairwise bounds. Lemma 3.1 provides a convenient and inexpensive way to compute the pairwise bounds $l(C_i, C_j)$ and $u(C_i, C_j)$, which are required to compute the multivariate correlation bounds with Theorems 3.2 and 3.3. In high-dimensional data, the bounds of Lemma 3.1 tend to be pessimistic, as they always account for the worst theoretical case. In the example of Fig. 2(c), theoretical bounds account for the case that hypothetical vectors (depicted in red) are located on the clusters' edges such that there exist vector pairs that are as close and as far away from each other as possible, given the position of the cluster centroids (black) and cluster radii. The closest possible pair of hypothetical vectors yields the upper bound, whereas the farthest pair yields the lower bound of the correlation.

An alternative 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 time, we compute the correlations between all pairs of vectors 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 the two values $l(C_i, C_j)$ 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. These values satisfy the definition of l and u, and are at least as tight as the bounds of Lemma 3.1. All computed l and u are also cached and reused every time Alg. 1 applies Theorems 3.2 and 3.3 for different cluster combinations. To distinguish between these new bounds derived empirically and the bounds derived by Lemma 3.1, hereafter we will refer to the new bounds as the empirical pairwise bounds.

There is a clear tradeoff between the cost of computing the empirical pairwise bounds, 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 a factor of two 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%, which empowered CD to reach to decisive combinations faster. The total execution time of the algorithm with empirical bounds was typically an order of magnitude less than the time with the theoretical bounds.

3.1.3 Exploiting additional user constraints. The irreducibility and minimum jump constraints (Section 2) can be easily integrated in the algorithm. 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. Enumeration of all subsets would cost $O(2^{|\mathcal{S}_l|+|\mathcal{S}_r|})$. To avoid this cost, during the execution of Alg. 1 we consider only the pairwise correlations between any two clusters $C_l \in \mathcal{S}_l$ and $C_r \in \mathcal{S}_r$. Precisely, we use the lower bound of the pairwise correlations between C_l and 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 irreducibility as it will always contain a pair of vectors with correlation higher than τ. 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.

Considering only the pairwise correlations during the pruning process can result in a notable performance increase, depending on the parameter settings. However, it may lead to inclusion of answers that do not satisfy the constraints. Therefore, the answer set of our algorithm is likely to be a superset of the true answers. Before returning the final answers to the user, the algorithm makes a final pass over the answers, and removes the ones that do not fully satisfy the constraints by checking the correlations of all subsets of an answer. Since the number of answers is typically in the order of a few tens to thousands, this final pass takes negligible time.

3.2 Top-k queries

When exploring new datasets with unknown characteristics, it can be daunting for the user to decide on a threshold τ. Setting a too high threshold for the dataset may lead to no answers, whereas a very low τ can result to millions of answers, and a decrease in performance. Top-k queries address this issue by allowing the user to set the desired number of results instead of τ. The answer then

includes the k combinations of vectors with the highest correlation that satisfy the correlation pattern and additional constraints.

Assuming an oracle that can predict the τ that would yield k results, the top-k queries could be transformed to threshold queries and answered with the standard CD algorithm. Since such an oracle is impossible, many top-k algorithms (e.g., Fagin's threshold algorithm [7]) 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-k variant of CD (see Alg. 2) is an exact algorithm and 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 highest k'th pairwise correlation. Since all pairwise correlations are anyway computed for the empirical bounds, this causes zero additional cost.

The second technique is an optimistic refinement of the upper bound, which aims to prioritize the combinations that will give 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 $UB_{\text{shrunk}} = (1-\gamma)\frac{UB+LB}{2} + \gamma UB$, where γ 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 k solutions from $\mathcal{R}$ with the highest correlation are kept (Alg. 2, lines 3-4). The value of τ is then updated accordingly to the minimum correlation in $\mathcal{R}$ (line 5). Otherwise, if UB_{shrunk} is greater than the current τ, analogous to Alg. 1, we recursively break the cluster to smaller clusters, until we get decisive bounds (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} surpasses τ. Values of γ^* are by definition between 0 and 1. 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 16-19), the algorithm processes the buckets one by one, starting from the first one, invoking the threshold query algorithm on each of its cluster combinations (Alg. 1) and updating the running τ after every bucket. Since the value of τ continuously increases, most combinations after the first few buckets will be filtered without needing many cluster splits.

Top-k as a basis for a progressive algorithm for 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 $k \to \infty$. 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 Alg. 1, or stop the algorithm when she is satisfied with the number of answers. We will evaluate the progressive nature of CD in Section 5.

Algorithm 2: Top-k-Query($\mathcal{S}_l$, $\mathcal{S}_r$, $Corr$, τ, k, γ, 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 τ, desired output set size k, shrinkfactor γ, buckets B.

```
// Phase 1
```

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:k]
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-k-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:k]
19 $\tau \leftarrow \min_{(X,Y)\in\mathcal{R}} Corr(X, Y)$

4 DETECTION OF MULTIPLE CORRELATIONS ON STREAMING DATA

One way to detect multiple correlations on streams is to re-run CD at regular intervals, each time on a new snapshot of the time series, e.g., the sliding window of the last 1000 seconds for each time series. However, as we will demonstrate later with experiments, this is not always the best approach. Our streaming algorithm, called CDStream, builds on top of CD such that it maintains the solution as new data arrive. Currently, CDStream works with the multiple correlation measure only; efficient support for the multipole measure is part of our ongoing work.

CDStream relies on two observations to increase the performance of CD. First, most updates do not lead to significant updates on the final result. Second, in most real-world scenarios, each of the monitored 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.

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 $C(\mathbf{v})$. The combinations that need to be checked after the update of $\mathbf{v}$ are only the ones that involve a cluster from $C(\mathbf{v})$ and are decisive – either positive or negative, since these are the ones that produce or discard results. The final result remains correct if these combinations are still decisive positive/negative.

The direct, yet expensive, approach to test if a combination is still decisive positive/negative requires recomputing the combination's bounds according to Theorem 3.2. CDStream adopts a faster method. Observe that, even for combinations with three or more clusters, the bounds in both theorems are determined by the minimum and maximum *pairwise* correlations between all involved clusters, denoted as $l(C_i, C_j)$ and $u(C_i, C_j)$ for any pair of clusters C_i and C_j. Therefore, any update that does not change $l(C_i, C_j)$ and $u(C_i, C_j)$ will not 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 extrema pair** 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, the bounds of Theorem 3.2, and the final solution accordingly.

Key to the performance of CDStream is an index that enables the algorithm to quickly locate the extrema pairs that are potentially affected by any update. Precisely, the index maps each vector $\mathbf{v}$ to a list of all decisive combinations (both negative and positive) that involve any cluster from $C(\mathbf{v})$.

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 $\langle \mathbf{b}, \mathbf{f} \rangle$. The first two of the five combinations both involve the pair of clusters C_2 and C_3. C_3 is the cluster that contains $\mathbf{c}$ and has $\mathbf{f}$ as a extremum, whereas C_2 has $\mathbf{b}$ as an extremum. 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, the vector $\mathbf{c}$). In our example, the first two combinations both involve the same extrema pair, and both have C_2 as the second cluster. The remaining three combinations under extrema pair $\langle \mathbf{b}, \mathbf{f} \rangle$ contain cluster C_8 instead as a second cluster, and therefore are placed in a different sub-group. We will refer to these clusters for the same extrema pair as the **extrema pair clusters**.

The described index is used to support a triggering functionality, which allows us to quickly locate and verify the extrema pairs related to each update. An update of any vector $\mathbf{v}$ is processed as follows. First, the index is used to retrieve the information related to $\mathbf{v}$. The algorithm iterates over the respective extrema pairs to verify that these did not change or move, despite the recent update of $\mathbf{v}$. Precisely, for each minimum extrema pair with correlation $\rho_{\min}$, it checks if the correlation of $\mathbf{v}$ with all points belonging to the second cluster is still at least equal to $\rho_{\min}$, whereas for each maximum extrema it verifies that all correlations of $\mathbf{v}$ remain less than $\rho_{\max}$. If this is still the case, all decisive combinations are still correct and do not need to be checked one-by-one. If, on the other hand, an extrema pair is invalidated (e.g., if the updated correlation of $\mathbf{v}$ with a vector from the opposite cluster becomes less than $\rho_{\min}$), the respective decisive combinations are checked and their bounds are recomputed and updated.

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

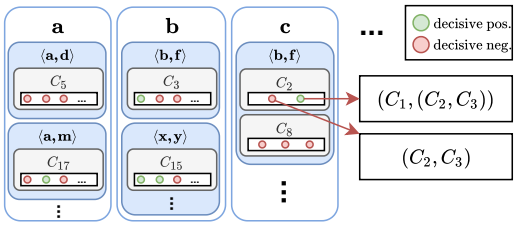


Figure 3: Visualization of the decisive combination index

– otherwise the other clusters could not contain the same extrema vector. To avoid double work, 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. 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 3.1. Furthermore, since the decisive combinations of the largest cluster were updated, the second, third, etc. largest extrema pair clusters are tested recursively. The process can stop 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 performance. To understand the reason, consider that each pair of clusters may appear in many decisive combinations. In the example of Fig. 2(c), assuming that $l_{\max} + r_{\max} = 3$ and the **mc** measure is used, 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 $n = 1000$.

Batching. CDStream also 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 performance with freshness of the answers. A larger batch size increases performance and throughput, as it allows CDStream to process many updates at once, 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-k queries with CDStream

We slightly modify the use of the index of Fig. 3 to handle user constraints and top-k queries. Notice again that the index supports a triggering functionality – a quick lookup and verification of the conditions that need to hold after an update on one or more vectors. In the absence of additional query constraints, these conditions concern only the validity of the extrema pairs for each pair of clusters in the decisive combinations. The minimum jump and irreducibility constraints are modeled in a similar fashion, and added in the index.

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 will exclude (X, Y) from the results if (X', Y') is in. We need to detect two 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 (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 condition (a), when will an increase of $Corr(X', Y')$ cause the following condition to hold: $Corr(X', Y') > \tau$) 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 that of the irreducibility contraint. The following cases need to be considered: (a) (X, Y) needs to be removed from the result set because $Corr(X', Y') > \tau$ and $Corr(X', Y') + \delta > Corr(X, Y)$, 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-k queries Recall that CDStream is initialized with the result of CD. For a top-k query, CDStream queries CD for a slightly larger number of results $k' = b * k$, where b 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 k, the true top-k results will always have a correlation higher than τ and will therefore be a subset of the top-k' results maintained by the algorithm. Before returning the results to the user, the top-k' correlations are sorted on correlation, and the top-k are returned to the user.

Scaling factor b controls the tradeoff between the robustness of the streaming algorithm for top-k queries, and its efficiency. A $b = 1$ may lead to the situation that, due to an update, fewer than k results exist with correlation greater than or equal to τ. CDStream will then fail to retrieve enough results, and resort 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-k 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 the streaming algorithm.

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 each approach for handling each 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 uses linear regression to model the relationship between the number of updates, and the execution time of each algorithm. Precisely, CDHybrid starts with a brief training period, where it tests out the performance of the two algorithms. During this period, it collects statistics on the observed arrival count and execution time. Simple linear regression is then used to extract a linear relationship between execution time and the observed number of updates. Note that the coefficients of a simple linear regression model can be computed in relation to the mean, variance, and covariance of the two variables, which can be maintained in constant time and space. Therefore, the process of updating the regression model continues after the training phase and takes constant time and space. The process of switching from one algorithm to the other is 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 producing 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 translated 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 for recomputation 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_1C_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, and (b) to compare them with an exhaustive search baseline, and with CoMEtExtended, the state-of-the-art algorithm for multivariate correlations [2].

Hardware, implementations. All experiments were executed on a server equipped with a 24-cores Intel Xeon Platinum 8260 Processor, and 400GB RAM. For the baseline method CoMEtExtended, we used the original implementation, which was kindly provided by the authors of [2]. We configured their implementation such

that it can use all available cores of our server. Our implementation of CD used the public-domain Jama library[3] for computing the eigenvalues, which were necessary for the multipoles measure.

Exhaustive search baseline. The multi-threaded exhaustive search baseline took extremely long, confirming the results of [2]. Indicatively, for **mp**(4) on a dataset of 1440 vectors (fMRI), the baseline took 40184 seconds to complete (our algorithm required 450 to 1000 seconds, depending on the parameters), whereas for **mc**(1, 3) it took 3232 seconds (our algorithm took 26.8 seconds). Notice that this baseline already cached and reused the pairwise correlations to speed-up calculations. Therefore, we will not report detailed results of the exhaustive search baseline.

Datasets. Several real-world datasets from different domains were used for the evaluation:[4]

• **Stocks.** Closing prices of 1596 stocks, covering a period from April 1, 2020 to May 12, 2020. Each stock had its own update frequency, ranging from one to 10 minutes. All prices were normalized with log-return normalization, as is standard in finance. Standard interpolation was used to cover missing values. Since correlations assume synchronized arrivals, all time series were resampled to 5 minutes for CD (leading to 9103 observations), and missing values were filled by interpolation. For CDStream, the time series were kept at the original update frequency. We again used interpolation to fill missing values, which was required for synchronizing the updates. The stock names are available in our github. The time series for these stocks can be downloaded from Yahoo!Finance.

• **fMRI.** Functional MRI data of a participant watching a movie.[5] The data was 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. The constant-value time series were removed, as these could not contribute to interesting correlations. This led to a total of 237, 509, 1440, 3152, and 9700 time series respectively, each with 5470 updates, covering a period of ~1.5 hours. Unless otherwise mentioned, the reported results will correspond to the 4x4x4 resolution.

• **Other datasets.** Dataset SLP [26] includes monthly Sea Level Pressure readings, for 171 locations. Each time series has 108 observations. We use the pre-processed SLP from [2]. Dataset ISD [24] contains hourly weather sensor data (temperature and sea-level pressure) between 2000-2005. Pre-processing involved removing the sensors with interruptions longer than 5 days, and resampling of the readings to regular 6-hours intervals. Missing values imputation was handled by forward-filling. The end dataset included 5713 time series with temperature data, and 2927 with sea-level pressure data, each with 8760 readings. CRYPTO [4] contained 3-hour closing prices of 7075 crypto-currencies, each with 713 observations, covering a period from April 14, 2021 to July 13. Pre-processing was identical to Stocks dataset.

Previous works already demonstrated the usefulness of discovering correlations and/or multivariate correlations, on many of the above data types, e.g., [1, 2, 16]. In the following, we will be showing detailed results for Stocks and fMRI, and indicative results for the other datasets. The reported results correspond to averages after 10 repetitions. Unless otherwise mentioned, a batch size of 50 and a sliding window size of 2000 were used in CDStream.

5.1 CD on static data

We start by examining the performance of CD on threshold and top-k queries, with and without constraints, and compare it to the state-of-the-art.

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 **mc**(2, 2) with different constraints, whereas the right Y axis corresponds to **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-k variant 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 – the upper bounds derived by Theorems 3.2 and 3.3 will be below τ more often, leading to less recursions.

Also note that CD is noticeably faster for **mc** compared to **mp**. Recall that computing the bounds for **mp** (Theorem 3.3) requires computing the eigenvalues of a matrix, which comes with a complexity of $O((l_{\max} + r_{\max})^3)$ in our current implementation. Furthermore, in the absence of additional constraints, **mp** usually leads to higher correlation values, which increase the number of answers drastically, and reduce the pruning power of the algorithm.

Table 1 presents the execution time of CD and size of the result set, 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 fMRI, the search space size grows 5 orders of magnitude between **mc**(1, 2) and **mc**(1, 4). The respective growth in execution time is two orders of magnitude less.

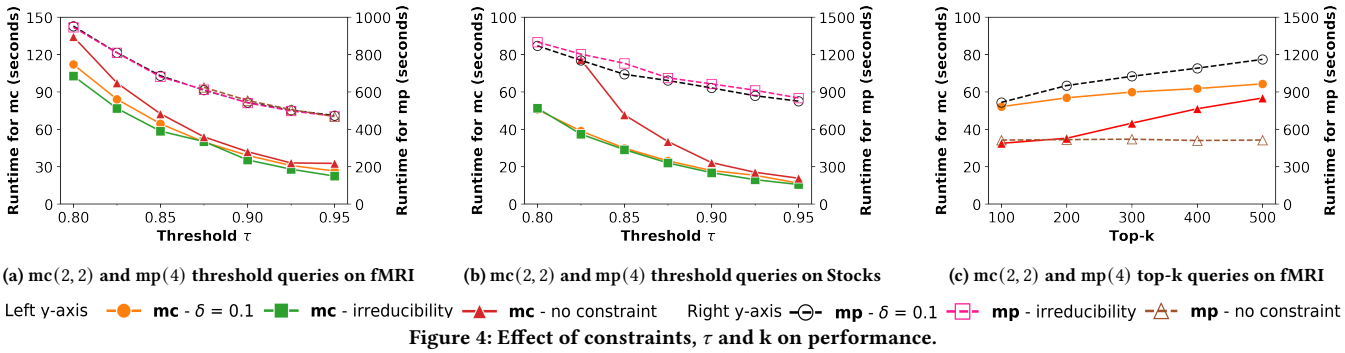
Our next experiment was configured to assess the effect of the dataset size on the efficiency of CD. For this experiment, we varied the resolution of the fMRI data by mean-pooling the activity level of a varying number of voxels, from cubes of size 2x2x2 to 8x8x8 voxels. This yielded datasets with 237 to 9700 time series. Figure 5a plots the runtime for the different dataset sizes, for **mc**(1, 3) and **mp**(3). The figure also includes the number of possible combinations for each correlation pattern. We again see that execution time for both correlation patterns grows at a much slower rate compared to the size of the search space, and the difference between the two increases with the dataset size. This discrepancy is attributed to the effective pruning provided by Theorems 3.2 and 3.3, which empowers CD to avoid exhaustively testing all combinations.

Our next experiment focused on comparing CD with CoMEtExtended. Notice that CoMEtExtended's goal differs slightly from our problem statement. CoMEtExtended aims to find only *maximal* sets that exhibit a strong multipole correlation, whereas CD finds *all* sets (up to a specified cardinality) that are strongly correlated.

[3] https://math.nist.gov/javanumerics/jama

[4] At the time of submission, all datasets are available online, at the owners' webpages. Since we do not own the datasets, we include details for their pre-processing in github.

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



(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-k queries on fMRI**

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

Furthermore, CoMEtExtended is an approximate algorithm. Its recall (and performance) can be controlled by parameter ρ_{CE}, which takes values between -1 and 1. Values around 0 offer a reasonable tradeoff between performance and recall [2]. In contrast, CD is exact, and therefore always retrieves all answers. We compared the two algorithms by examining their recall rates and runtimes. To ensure a fair comparison for CoMEtExtended, we also added all subsets of the sets returned by CoMEtExtended in its answer, thereby increasing its recall. This last step was not included in the runtime of CoMETExtended, such that 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) and for the same parameters used in [2]. Notice that we only consider the **mp** measure, since CoMEtExtended does not support **mc**. We see that CD is consistently faster than CoMEtExtended – up to an order of magnitude – and often returns significantly more results. Indicatively, for **mp**(4), CD is one to two orders of magnitude faster than CoMEtExtended with $\rho_{\text{CE}} = 0$, whereas for $\rho_{\text{CE}} = 0.02$, the difference in performance is between two to three orders of magnitude. Notice that for queries with $\delta = 0.1$, CoMEtExtended found 281 sets of cardinality 6, and one set of cardinality 7 ($\rho_{\text{CE}} = 0.02, \tau = 0.4$). Naturally, these were not discovered by CD, as the queries specified $l_{\max} = 5$ at most, prioritizing the simpler and more interpretable results.

We also evaluated the progressive nature of CD – how much time the algorithm requires to find a large portion of results. We modified our code such that it prints the number of retrieved results at different time points. We tested the code for patterns **mc**$(1,4)$ and **mp**(4) that take significant time to complete, since these are the ones that would mostly benefit from a progressive algorithm. Figure 5b plots the number of results returned by the algorithm, as a function of time, for Stocks dataset. We see that CD retrieves around half of the results in the first few seconds, and in around 10% of the time it reaches to around 80% recall. As expected, less complex correlation patterns were answered much faster.

5.1.2 Top-k queries. Fig. 4c plots the execution time of CD for different values of k with the fMRI dataset – the results with Stocks were qualitatively very similar. We see that a decrease of k typically leads to increased efficiency. A low value of k helps the algorithm to increase the running threshold τ faster (lines 4 and 18 of Alg. 2), leading to more aggressive pruning when Alg. 1 is invoked. Note that this behavior is less noticeable for **mp**(4) with no constraints. This can be attributed to the correlation values in the result set. Indicatively, for this query the lowest correlation in the result set

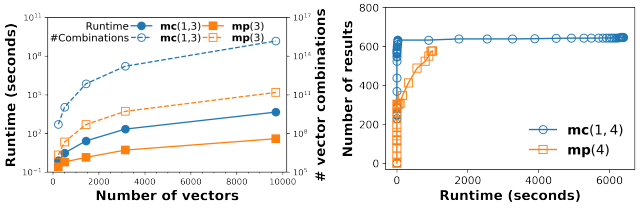


Figure 5: (a) Running time and number of vector combinations for different resolutions of the fMRI dataset, with $\tau = 0.9$ and no constraints. (b) Number of recovered results during the execution of mc$(1,4)$ and mp(4), with $\tau = 0.9, \delta = 0.05$ on the Stocks dataset.

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 correlation from 0.82 (top-100) to 0.78 (top-500), explaining why the effect of k on performance is significant.

Table 2 shows the execution times of CD for indicative correlation patterns on the other datasets. The insights are consistent to the ones from the experiments with fMRI and Stocks. As expected, dataset size and the complexity of the correlation pattern influence the execution time, and **mp** correlation patterns take more time compared to the **mc** patterns with an equal number of vectors.

5.2 Evaluation with streaming data

The second series of experiments was designed to evaluate the performance of CDStream. To generate the streams from our datasets, we used the arrival times that were already included in the data. Due to page constraints, our discussion will mainly focus on the Stocks dataset; results for the other datasets are shown only when they offer new insights.

For the first experiment we configured CDStream to use a batch size of 50 updates, and timed the processing of each batch. Figure 6a presents the mean processing time per batch, for subsets of the Stocks dataset ranging from 200 to 1000 randomly chosen vectors. For comparison purposes, the figure also includes the time required for executing CD at the end of each batch. We see that CDStream is significantly more efficient than CD for small correlation patterns, requiring a few milliseconds – an order of magnitude less time compared to CD. Also note that, even though the number of comparisons increases at a combinatorial rate with the number of vectors (cf. Fig. 5a), execution time grows at a substantially slower rate. This is attributed to the grouping technique in the index of CDStream, which effectively reduces the work for processing each

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

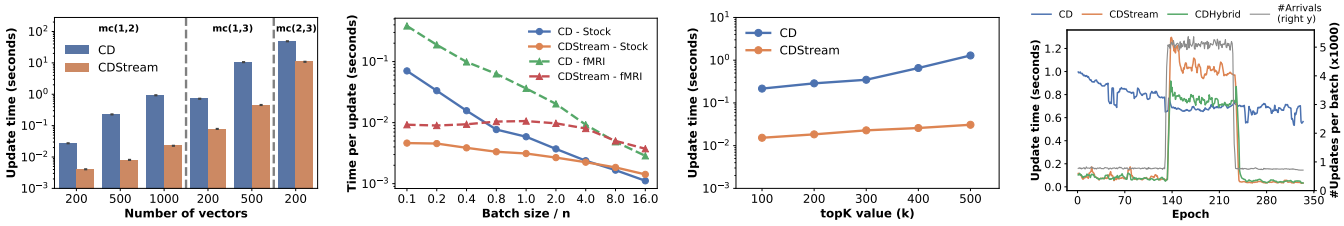
Table 1: CD with different queries

	ISD (temp.)		ISD (sea-level pr.)		Crypto	
	Time	Min corr	Time	Min corr	Time	Min corr
mc(1, 2)	13.0	0.991	3.3	0.998	6.2	0.906
mc(1, 3)	60.9	0.993	10.3	0.998	501.0	0.936
mc(2, 2)	174.2	0.995	34.9	0.999	303.1	0.942
mp(3)	22.2	0.992	8.4	0.999	30.2	0.938
mp(4)	7223.6	0.993	2952.2	0.999	12462.7	0.952

Table 2: Top-100 queries, no-constraints, on ISD and Crypto datasets

δ/τ	CD			CDStream		
	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

Table 3: Effect of τ and δ on CD and CDStream for streaming data



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

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

(c) Effect of k, with $\delta = 0.05$

(d) Efficiency of CDHybrid over time.

Figure 6: Effect of query parameters on performance of CDStream

	CoMEtExtended						Correlation Detective			
τ, δ	$\rho_{CE} = 0$		$\rho_{CE} = 0.01$		$\rho_{CE} = 0.02$		**mp**(4)		**mp**(5)	
	time	#res.	time	#res.	time	#res.	time	#res.	time	#res.
0.4, 0.1	604	62663	1318	67110	3530	70921	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 4: Comparison of CoMEtExtended and Correlation Detective: running time (seconds) and number of retrieved results

update. For larger correlation patterns, such as **mc**(2, 3), CDStream has performance comparable to CD.

Figure 6c plots the average processing time per batch for top-k query **mc**(1, 2), with different k values. The results correspond to the Stocks dataset with 1000 stocks. We see that processing time for both algorithms increases with k. In CD, execution time grows almost linearly with k (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 difference in performance is that CDStream only maintains the top-k solutions, already having a good estimate for the threshold of the top-k highest correlation from previous runs, whereas CD starts each run always from scratch.

Table 3 presents the effect of τ and constraints (minimum jump and irreducibility) on CDStream's performance. We see that the constraints have a negligible effect on the algorithm's performance, since they only slightly increase 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 the 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 offers a tunable tradeoff between throughput and update rate of the results: increasing the batch size increases the efficiency, but reduces the freshness of the results. This happens because a growing batch size increases the probability that some vectors receive multiple updates in the same batch. In this case, at the end of the batch, both algorithms will process only the latest values for each vector, ignoring the intermediary updates. Therefore, doubling the batch size roughly halves the time required for processing a single update. We also see that, as the batch size grows, performance of CD approaches performance of CDStream. A large batch containing many significant updates defeats the purpose of maintaining the solution with CDStream. For Stocks, processing time for the two algorithms crosses at a batch size $4 * n$, whereas for fMRI, crossing happens at batch size $8 * n$. This discrepancy is attributed to the core properties of the datasets, i.e., the inherent distributions and magnitude of updates.

Our final experiment examines the ability of CDHybrid to switch between CD and CDStream based on the stream properties. For this experiment, we use a time-based batch size of 1 minute, which allows us to vary the number of updates per batch. We simulated 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 (rolling average for the last 5 batches) throughout time, for processing Stocks with CD, CDStream, and CDHybrid. Each epoch corresponds to one batch. The right Y axis shows the number of arrivals within each batch. We observe that CDHybrid quickly switches to the best method. In the few epochs following a switch between algorithms, the cost of CDHybrid has a small peak that is attributed to the initialization cost of the algorithms. For the case of switching back to CDStream (epoch 240), the additional cost for updating the outdated index is also very 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.

6 CONCLUSIONS

We considered the problem of detecting high multivariate correlations, using two correlation measures, and applying 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. All algorithms rely on novel theoretical results, which enable us to bound multivariate correlations between large sets of vectors. A thorough experimental evaluation using 5 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] CoinGecko. 2021. CoinGecko API. https://www.coingecko.com/en/api. Accessed: 2021-07-13.

[5] 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.

[6] 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.

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

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

[9] 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.

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

[11] H. Hasan Örkcü. 2013. Subset selection in multiple linear regression models: A hybrid of genetic and simulated annealing algorithms. *Appl. Math. Comput.* 219, 23 (2013), 11018–11028.

[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. *Canonical Correlation Analysis*. Springer, 321–330.

[14] L. Kolotilina. 2000. A generalization of Weyl's inequalities with implications. *Journal of Mathematical Sciences* 101 (09 2000), 3255–3260.

[15] 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.

[16] 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.

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

[18] 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).

[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] Douglas Montgomery. 2011. *Applied statistics and probability for engineers*. Wiley, Hoboken, NJ.

[21] 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.

[22] 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.

[23] 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.

[24] National Oceanic and Atmospheric Administration. 2021. NOAA Integrated Surface Dataset (Global). https://www.ncei.noaa.gov/access/search/dataset-search. Accessed: 2021-07-30.

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

[26] 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.

[27] 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*.

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

[29] 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.

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

[31] 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.

[32] 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.

[33] 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.

[34] 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.

7 APPENDICES

7.1 Proof of Lemma 3.1

Let d be the length of $\mathbf{x}$ and $\mathbf{y}$. Note that $\rho(\mathbf{x},\mathbf{y}) = \frac{\mathbf{x}^T \cdot \mathbf{y}}{d \cdot \sigma(\mathbf{x}) \cdot \sigma(\mathbf{y})}$. Since $\mathbf{x}$ and $\mathbf{y}$ are z-normalized, $\sigma(\mathbf{x}) = \frac{||\mathbf{x}||_2}{\sqrt{d}}$, and similar for $\mathbf{y}$. Therefore, $\rho(\mathbf{x},\mathbf{y}) = \frac{\mathbf{x}\cdot\mathbf{y}}{||\mathbf{x}||_2 \cdot ||\mathbf{y}||_2} = cos(\theta_{\mathbf{x},\mathbf{y}})$.

Consider $\theta_{\mathbf{u_1},\mathbf{u_2}}$, with $\mathbf{u_1}$ and $\mathbf{u_2}$ as defined in the lemma. Then, from the triangle inequality:

$$\theta_{\mathbf{v_1},\mathbf{v_2}} - \theta_1 - \theta_2 \leq \theta_{\mathbf{u_1},\mathbf{u_2}} \leq \theta_{\mathbf{v_1},\mathbf{v_2}} + \theta_1 + \theta_2$$

Under the convention that all angles between two vectors are in $[0,\pi]$, we can rewrite this as

$$\max(0, \theta_{\mathbf{v_1},\mathbf{v_2}} - \theta_1 - \theta_2) \leq \theta_{\mathbf{u_1},\mathbf{u_2}} \leq \min(\pi, \theta_{\mathbf{v_1},\mathbf{v_2}} + \theta_1 + \theta_2)$$

Since cos is a monotonically decreasing function on $[0,\pi]$, and $\cos(\theta_{\mathbf{u_1},\mathbf{u_2}}) = \rho(\mathbf{u_1},\mathbf{u_2})$, we get the final result:

$$\cos(\min(\pi, \theta_{\mathbf{v_1},\mathbf{v_2}} + \theta_1 + \theta_2)) \leq \rho(\mathbf{u_1},\mathbf{u_2}) \leq \cos(\max(0, \theta_{\mathbf{v_1},\mathbf{v_2}} - \theta_1 - \theta_2))$$

7.2 Proof of Theorem 3.2

The **mc** correlation function can be rewritten as follows:

$$\begin{aligned}
\mathbf{mc}(X,Y) = \rho(\mathbf{Avg(X)}, \mathbf{Avg(Y)}) &= \frac{\sum_{i=1}^{d} Avg(X)_i \cdot Avg(Y)_i}{d \cdot \sigma(\mathbf{Avg(X)}) \cdot \sigma(\mathbf{Avg(Y)})} \\
&= \frac{\sum_{i=1}^{d} \left(\sum_{\mathbf{x}\in X} x_i\right) \cdot \left(\sum_{\mathbf{y}\in Y} y_i\right)}{d \cdot \sigma\left(\sum_{\mathbf{x}\in X} \mathbf{x}\right) \cdot \sigma\left(\sum_{\mathbf{y}\in Y} \mathbf{y}\right)} \\
&= \frac{\sum_{\mathbf{x}\in X} \sum_{\mathbf{y}\in Y} \frac{1}{d} \sum_{i=1}^{d} x_i \cdot y_i}{\sqrt{\sum_{\mathbf{i},\mathbf{j}\in X} cov(\mathbf{i},\mathbf{j})} \cdot \sqrt{\sum_{\mathbf{i},\mathbf{j}\in Y} cov(\mathbf{i},\mathbf{j})}} \\
&= \frac{\sum_{\mathbf{x}\in X} \sum_{\mathbf{y}\in Y} \rho(\mathbf{x},\mathbf{y})}{\sqrt{\sum_{\mathbf{i},\mathbf{j}\in X} \rho(\mathbf{i},\mathbf{j})} \cdot \sqrt{\sum_{\mathbf{i},\mathbf{j}\in Y} \rho(\mathbf{i},\mathbf{j})}}
\end{aligned}$$

Since $\rho(\cdot,\cdot)$ is undefined when one or both input vectors have standard deviation equal to zero, $\sigma(\mathbf{Avg(X)}) > 0$, $\sigma(\mathbf{Avg(Y)}) > 0$, and the denominator is always greater than zero.

As $\mathcal{S}$ is a set of clusters rather than fixed vectors, the correlations $\rho(\cdot,\cdot)$ are not fixed. However, these can be lower- and upper-bounded with $l(\cdot,\cdot)$ and $u(\cdot,\cdot)$ respectively. Then, the numerator is bounded as follows: $L(\mathcal{S}_\mathrm{l}, \mathcal{S}_\mathrm{r}) \leq \sum_{\mathbf{x}\in X, \mathbf{y}\in Y} \rho(\mathbf{x},\mathbf{y}) \leq U(\mathcal{S}_\mathrm{l}, \mathcal{S}_\mathrm{r})$

The denominator is bounded as follows: $\sqrt{L(\mathcal{S}_\mathrm{l}, \mathcal{S}_\mathrm{l})}\sqrt{L(\mathcal{S}_\mathrm{r}, \mathcal{S}_\mathrm{r})} \leq \sqrt{\sum_{\mathbf{i},\mathbf{j}\in \mathbf{X}} \rho(\mathbf{i},\mathbf{j})}\sqrt{\sum_{\mathbf{i},\mathbf{j}\in \mathbf{Y}} \rho(\mathbf{i},\mathbf{j})} \leq \sqrt{U(\mathcal{S}_\mathrm{l}, \mathcal{S}_\mathrm{l})}\sqrt{U(\mathcal{S}_\mathrm{r}, \mathcal{S}_\mathrm{r})}$

Since the denominator is always greater than zero:

$\max(0, \sqrt{L(\mathcal{S}_\mathrm{l}, \mathcal{S}_\mathrm{l})}\sqrt{L(\mathcal{S}_\mathrm{r}, \mathcal{S}_\mathrm{r})}) \leq \sqrt{\sum_{\mathbf{i},\mathbf{j}\in \mathbf{X}} \rho(\mathbf{i},\mathbf{j})}\sqrt{\sum_{\mathbf{i},\mathbf{j}\in \mathbf{Y}} \rho(\mathbf{i},\mathbf{j})} \leq \sqrt{U(\mathcal{S}_\mathrm{l}, \mathcal{S}_\mathrm{l})}\sqrt{U(\mathcal{S}_\mathrm{r}, \mathcal{S}_\mathrm{r})}$

Theorem 3.2 follows by distinguishing the three cases. □

7.3 Proof of Theorem 3.3

The multipole correlation $\mathbf{mp}(X)$ can be rewritten as [2]:

$$\mathbf{mp}(X) = 1 - \lambda_{min}(\mathbf{R})$$

where $\lambda_{min}(\mathbf{A})$ denotes the smallest eigenvalue of a matrix $\mathbf{A}$, and $\mathbf{R}$ is the correlation matrix of the vectors in X. As $\mathcal{S} = \{C_1, \ldots, C_n\}$ is a set of clusters, the elements of $\mathbf{R}$, r_{ij}, are not fixed. In what follows, let $\mathbf{R}$ denote any correlation matrix corresponding to a set of vectors X in which each vector $\mathbf{x_i} \in C_i$. Matrices $\mathbf{L}$ and $\mathbf{U}$ are constructed such that $\forall_{i,j\in\{0,\ldots,n\}} : l_{ij} \leq r_{ij} \leq u_{ij}$. Furthermore, for conciseness, let $\mathbf{M} = \frac{\mathbf{U}+\mathbf{L}}{2}$ and $\mathbf{E} = \mathbf{R} - \mathbf{M}$. Then:

$$\lambda_{min}(\mathbf{R}) = \lambda_{min}(\mathbf{M} + \mathbf{E})$$

Using Weyl's inequality [14], we can derive:

$$\begin{aligned}
|\lambda_{min}(\mathbf{M}+\mathbf{E}) - \lambda_{min}(\mathbf{M})| \leq ||\mathbf{E}||_2 \Leftrightarrow \\
|\lambda_{min}(\mathbf{R}) - \lambda_{min}(\mathbf{M})| \leq ||\mathbf{E}||_2
\end{aligned}$$

To complete the proof, it remains to be shown that $||\mathbf{E}||_2 \leq \frac{1}{2}||\mathbf{U}-\mathbf{L}||_2$. To this end, let $\mathbf{x}^* = \underset{||\mathbf{x}||_2=1}{argmax} ||\mathbf{E}\mathbf{x}||_2$, and let $\mathbf{y}$ denote the vector containing the absolute values of the elements in $\mathbf{x}^*$: $y_i = |x_i^*|$. Note that $||\mathbf{y}||_2 = 1$. Then:

$$\begin{aligned}
||\mathbf{E}||_2 &= \max_{||\mathbf{x}||_2=1} ||\mathbf{E}\mathbf{x}||_2 \\
&= ||\mathbf{E}\mathbf{x}^*||_2 \\
&= \sqrt{\sum_{i=1}^{p}\left(\sum_{j=1}^{p}\left(r_{ij} - \frac{u_{ij}+l_{ij}}{2}\right)x_j^*\right)^2} \\
&\leq \sqrt{\sum_{i=1}^{p}\left(\sum_{j=1}^{p}\left|r_{ij} - \frac{u_{ij}+l_{ij}}{2}\right|\left|x_j^*\right|\right)^2} \\
&\leq \sqrt{\sum_{i=1}^{p}\left(\sum_{j=1}^{p}\left|u_{ij} - \frac{u_{ij}+l_{ij}}{2}\right| y_j\right)^2} \\
&= \sqrt{\sum_{i=1}^{p}\left(\sum_{j=1}^{p}\left(\frac{u_{ij}-l_{ij}}{2}\right) y_j\right)^2} \\
&= ||\left(\frac{\mathbf{U}-\mathbf{L}}{2}\right)\mathbf{y}||_2 \\
&\leq \max_{||\mathbf{v}||_2=1} ||\left(\frac{\mathbf{U}-\mathbf{L}}{2}\right)\mathbf{v}||_2 \\
&= \frac{1}{2}||\mathbf{U}-\mathbf{L}||_2
\end{aligned}$$

□