Scalable Grid-based Computation of Kendall's τ Correlation

Nikolaos Koutroumanis
n.koutroumanis@athenarc.gr
Archimedes, Athena RC
Athens, Greece

Petros Karampas
petroskarampas697@gmail.com
TeamViewer
(Work done while at Univ. Ioannina)
Ioannina, Greece

Alexandros Karakasidis
a.karakasidis@uom.edu.gr
University of Macedonia
Thessaloniki, Greece

Nikos Mamoulis
nikos@cs.uoi.gr
University of Ioannina &
Archimedes, Athena RC
Ioannina, Greece

Panos Vassiliadis
pvassil@cs.uoi.gr
University of Ioannina
Ioannina, Greece

ABSTRACT

Computing the correlation of two attributes in a large dataset is an important problem, with many applications, including exploratory analytics and dimensionality reduction. Among the well-known correlation measures, Kendall's τ is the most robust one, as it is immune from parametric assumptions and outliers. On the other hand, computing Kendall's τ for large-scale data becomes challenging (i) due to the superlinear cost of the state-of-the-art algorithm and (ii) because all data need to be memory-resident for efficient processing. In this paper, we address the problem via a geometric approach that partitions the data in the cells of a grid, and exploits the relative position of the cells to compute correlation information en masse. Our approach facilitates parallel and distributed computation of Kendall's correlation; we propose a scalable algorithm in this direction. Finally, we propose an efficient approximate algorithm with a provable error bound, which derives accurate results by a single pass over the grid statistics. Our experimental evaluation demonstrates the efficiency and scalability of our grid-based techniques compared to the state-of-the-art algorithm.

PVLDB Reference Format:
Nikolaos Koutroumanis, Petros Karampas, Alexandros Karakasidis, Nikos Mamoulis, and Panos Vassiliadis. Scalable Grid-based Computation of Kendall's τ Correlation. PVLDB, 19(5): 876 - 888, 2026.
doi:10.14778/3796195.3796202

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/DAINTINESS-Group/KendallTauForBigData.

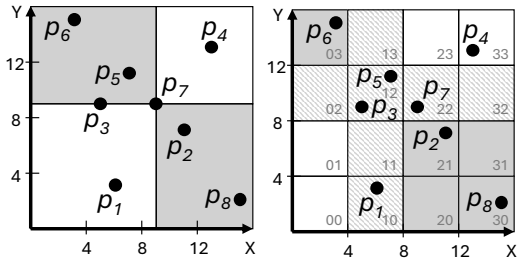
1 INTRODUCTION

Correlation analysis measures the nature and strength of the relationship between two attributes in a relation. Identifying correlated attributes enriches the knowledge and deep understanding on interdependencies between attributes in multivariate data. Among other tasks, correlation analysis facilitates feature selection and regression estimation. Correlated variables do not need to be included together in the same exploratory analysis. In addition, finding correlated pairs or sets of attributes facilitates dimensionality reduction [3]. Correlation analysis has been used in modeling business processes [1], time-series mining and cleaning [25, 32, 41], indexing [18, 37], query optimization [44], and column compression [23].

Measures of correlation. A popular correlation measure is Pearson's r coefficient [7, 11, 24, 26, 33, 38, 42]. Still, the semantics of Pearson's r assess only *linear* relationships, failing to accurately identify other forms of correlation (e.g., polynomial or logarithmic). Also, r is significantly sensitive to outliers. Spearman's ρ and Kendall's τ_b correlation coefficients address these shortcomings, by measuring *rank correlation*, i.e., the ordinal association of two variables; ρ and τ_b are both more expensive to compute than r, having $O(n \log n)$ cost. They are considered quite similar and typically produce values that are close to each other [31]. Their difference has to do with their interpretation: Kendall's correlation tests the hypothesis that the two variables are statistically independent and essentially represents a probability. Spearman's ρ measures the proportion of variability accounted for, by the correlation of the two variables. Furthermore, ρ can be biased for cases where there are many tied ranks, making it less robust than τ_b. For a comparison between Spearman's and Kendall's correlation, we can resort to Xu et al. [39], who concluded after a thorough theoretical and experimental analysis that ρ is preferred for *small samples with weak correlations*, while τ_b is preferred for *strong correlations, large samples, and the presence of noise* (i.e., outliers) – practically meaning deviation from a normal distribution.

Kendall's τ_b. In this paper, we focus on Kendall's τ_b, which is more robust than Pearson's and Spearman's methods, being immune to parametric assumptions and outliers. Kendall's τ_b has been used across various domains, including computational linguistics [21], system analytics [8], environmental sciences [22], and healthcare [5, 6, 15]. Intuitively, the coefficient shows how well the ranks of one variable agree with the ranks of another variable, assessing whether they tend to rise or fall together in the same way. Figure 1a shows a set of observations of two variables X and Y, each mapped to a point in the 2D space. A pair of observations (e.g., p_1 and p_7) is *concordant* when the order relation between their X and Y values is the same (e.g., $p_1.X < p_7.X$ and $p_1.Y < p_7.Y$); if the order relation

Proceedings of the VLDB Endowment, Vol. 19, No. 5 ISSN 2150-8097.
doi:10.14778/3796195.3796202



(a) Pairs that involve p_7 **(b) Grid partitioning**

Figure 1: A set of observations (as points).

is different, as in pair (p_5, p_7), the pair is *discordant*. The value of τ_b ranges from -1 to 1; τ_b is positive if the concordant pairs are more than the concordant ones; in the opposite case (as in Figure 1), τ_b is negative. A value close to zero indicates independent behavior for the two attributes. τ_b is dampened by the existence of tied values in the X or Y dimensions (see the definition in Section 2.1).

Contribution. Knight [19] suggested a variant of the merge-sort algorithm that can compute τ_b in $O(n \log n)$. This method naturally extends merge-sort, however, it relies on binary merging of sorted runs and hence it is not scalable for big data that do not fit in memory. In this paper, we propose a scalable geometric approach, which *partitions* the data to the cells of a grid that uses the relationships between cells for computing discordant and concordant pairs between groups of points that belong to different cells. The proposed approach not only delivers comparable or superior performance against Knight's algorithm, but it also is inherently parallelizable under the MapReduce paradigm, making it suitable for large datasets not fitting in the main memory of a single computer. Figure 1b shows an example of such a partitioning. We take advantage of the grid statistics to reduce comparisons for diagonally oriented cells. For example, cell 12 which includes p_3 and p_5 (2 observations), contributes $2 \times 2 = 4$ discordant pairs, since cells 20, 21, 30, and 31 collectively contain 2 points. We apply Knight's algorithm [19] to compute discordant pairs within the same cell and we propose a merge-join algorithm for pairs of cells that belong to the same row or column of the grid. By explicitly measuring ties and discordant pairs, we can infer the concordant ones, and thus the *exact* value of Kendall's τ_b. For big data, we propose a distributed version of our method that applies the MapReduce framework and scales gracefully. Finally, we suggest a single-pass algorithm over grid cell statistics that can compute the discordant pairs between cells not in the same row or column, which can also be adapted to a very fast algorithm for approximate Kendall's τ_b correlation computation with approximation guarantees.

After defining the problem in Section 2, we proceed with our contributions, which can be summarized as follows:

- In Section 3, we propose a geometric, partitioning-based and inherently parallelizable approach for the computation of Kendall's τ_b coefficient. Our method includes a set of grid-based processing modules collecting the necessary statistics for Kendalls' τ_b computation.
- In Section 4, we scale up and out Kendall's Tau computation by evenly splitting the required processing into equi-weighted jobs, which are assigned to a number of nodes. We present a MapReduce-based version of our method using Spark for scalable computation.
- In Section 5, we introduce an approximate algorithm of Kendall's τ_b, which avoids expensive comparisons and trades efficiency for accuracy.
- In Section 6, we experimentally evaluate the efficiency and scalability of our approach in both centralized and distributed environments.

We conclude the paper with a survey of related work and directions for future work.

2 PRELIMINARIES

In this section, we present a definition of Kendall's τ_b rank correlation measure and the state-of-the-art algorithm.

2.1 Kendall's Tau

Let (X, Y) be a binary matrix with n observations. Kendall's Tau is a family of non-parametric rank correlation coefficients [17] that measure the ordinal association between columns X and Y. It includes three main rank coefficient variants, Kendall's Tau-a (τ_a), Tau-b (τ_b) and Tau-c (τ_c), where τ_b is the most widely used due to its ability to account for tied ranks, providing a more precise measure of the rank correlation. Hence, in this work we focus exclusively on computing τ_b.

Kendall's τ_b classifies pairs of observations into *concordant*, *discordant*, *tied on X*, or *tied on Y*. A pair of observations (x_i, y_i) and (x_j, y_j) of the joint random variables X and Y are said to be concordant if the ranks for both elements agree; that is, if $x_i > x_j$ and $y_i > y_j$ or if $x_i < x_j$ and $y_i < y_j$. The observations are said to be discordant if $x_i > x_j$ and $y_i < y_j$ or if $x_i < x_j$ and $y_i > y_j$. If $x_i = x_j$ and $y_i \neq y_j$ the pair is tied on X and if $x_i \neq x_j$ and $y_i = y_j$ the pair is tied on Y. If two observations are identical, i.e. $x_i = x_j$ and $y_i = y_j$, then they are disregarded.

Definition 2.1. [Kendall's Tau-b (τ_b)] Assuming a relation (X, Y) with X and Y being ordinal variables, number C of concordant pairs, number D of discordant pairs, number of pairs T_x tied at variable X and number of pairs T_y tied at variable Y, the *Kendall* τ_b correlation of X and Y is defined as:

$$\tau_b(X, Y) = \frac{C-D}{\sqrt{(C+D+T_x)(C+D+T_y)}}$$

Kendall's $\tau_b(X, Y)$ ranges in $[-1, 1]$, with 1 indicating that X and Y have identical rankings, -1 inverse rankings, and, 0 indicating independent behavior for X and Y.

2.2 Knight's algorithm

To compute $\tau_b(X, Y)$, a naive approach applies comparisons between all $\frac{n(n-1)}{2}$ pairs of observations. Knignt [19] noted that, after sorting the binary matrix by X, the discordant pairs can be obtained by counting the number of swaps that the BubbleSort algorithm needs to sort the Y column of the matrix. Knight's algorithm adapts the MergeSort algorithm to count these swaps in $O(n \log n)$ time.

Algorithm 1: Knight's Algorithm.

Input: A 2D array (X, Y) of n observations
Output: $\tau_b(X, Y)$

Function knight((X, Y)):
  $dis \leftarrow 0; tiX \leftarrow 0; tiY \leftarrow 0; tiXY \leftarrow 0$
  Sort (X, Y) *on* X, Y
  With a single pass over X*-sorted* (X, Y)*, compute number* tiX *of tied pairs on* X *and number* $tiXY$ *of tied pairs on* X *and* Y
  $\{dis, (X, Y)\} \leftarrow$ mergeSortDis((X, Y))
  With a single pass over Y*-sorted* (X, Y)*, compute number* tiY *of tied pairs on* Y
  $con = n \cdot (n-1)/2 - dis - tiX - tiY - tiXY$
  return $(con - dis)/\sqrt{(con + dis + tiX)(con + dis + tiY)}$

Function mergeSortDis((X, Y)):
  if $|(X, Y)| = 1$ **then**
    return $\{0, (X, Y)\}$
  else
    Split (X, Y) *into two halves:* $(X, Y)_L$ *and* $(X, Y)_R$
    $\{swaps_L, (X, Y)_{LS}\} \leftarrow$ mergeSortDis($(X, Y)_L$)
    $\{swaps_R, (X, Y)_{RS}\} \leftarrow$ mergeSortDis($(X, Y)_R$)
    $\{swaps_{LR}, (X, Y)_S\} \leftarrow$ mergeDis($(X, Y)_{RL}, (X, Y)_{RS}$)
    $totSwaps \leftarrow swaps_L + swaps_R + swaps_{LR}$
    return $\{totSwaps, (X, Y)_S\}$

Function mergeDis($(X, Y)_{LS}, (X, Y)_{RS}$):
  $i \leftarrow 1; j \leftarrow 1; swaps \leftarrow 0$
  $outArray = [\,]$
  while $i \leq |(X, Y)_{LS}|$ *and* $j \leq |(X, Y)_{RS}|$ **do**
    if $(X, Y)_{RS}[j].y < (X, Y)_{LS}[i].y$ **then**
      $outArray.append((X, Y)_{RS}[j])$
      $swaps \leftarrow swaps + |(X, Y)_{LS}| - i + 1$
      $j \leftarrow j + 1$
    else
      $outArray.append((X, Y)_{LS}[i])$
      $i \leftarrow i + 1$
  return $\{swaps, outArray\}$

Algorithm 1 shows the steps of Knight's approach, which also counts ties, as implemented in commons.apache.org. The algorithm first sorts the input array (X, Y) primarily by X and secondarily by Y. With one pass over the sorted array it can compute (i) the total number tiX of pairs which have identical X only and (ii) the total number $tiXY$ of pairs which are identical in both X and Y. Then, the algorithm runs an adapted version of MergeSort (function mergeSortDis), which sorts (X, Y) by Y and at the same time counts the total number of swaps that BubbleSort would require for such a sorting. This number is the same as the number dis of discordant pairs in (X, Y) [19]. Then, the resulting Y-sorted array is scanned to compute the total number tiY of pairs which have identical Y. The number con of concordant pairs can be computed by subtracting from the total number of pairs dis, tiX, tiY, and $tiXY$. Finally, τ_b is computed using Definition 2.1.

3 GRID-BASED τ_b COMPUTATION

In this section, we present our grid-based approach for the efficient computation of Kendall's τ_b rank correlation of two attributes. Figure 2 shows an overview of our approach. Algorithm 2 outlines the steps of our method. We first partition the observations using a grid and we sort the contents of each cell by X (Lines 2–4). Then, for each column i of the grid, we apply an *inter-cell merge join* for all pairs of cells $(c_{ij}, c_{ik}), k < j$ (function mergeSortSouth). This join module, which will be described in detail later, computes the discordant and tied-at-X pairs $(p_l, p_m), p_l \in c_{ij}, p_m \in c_{ik}$. Next, for each cell c, the algorithm computes all intra-cell discordant and tied pairs, using Knight's algorithm (Section 2). Running Knight's algorithm for a cell c results in sorting its content by Y. The next step is, for each column j of the grid, to apply an inter-cell merge join for all pairs of cells $(c_{kj}, c_{ij}), k < i$ (function mergeSortEast). Finally, the algorithm runs the discordantCells module, to compute for each cell c the discordant pairs from c and all cells south-east of c. As in Algorithm 1, the concordant pairs are computed by subtracting from the total number of pairs the tied pairs in any or both axes. We now elaborate on the mergeSortSouth, mergeSortEast, and discordantCells modules.

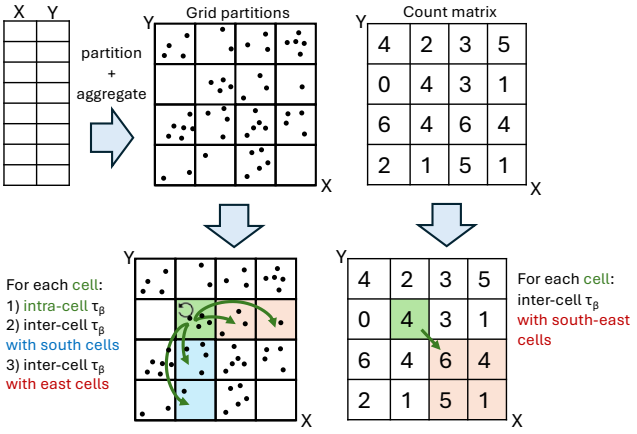


Figure 2: Overview of grid-based Kendall τ_b computation.

Algorithm 2: Grid-based τ_b computation.

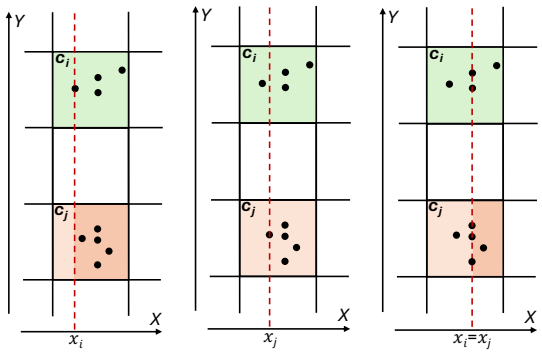
Input: A $m_x \times m_y$ grid G, a 2D array (X, Y) of n observations
Output: $\tau_b(X, Y)$

1 $dis \leftarrow 0; tiX \leftarrow 0; tiY \leftarrow 0; tiXY \leftarrow 0$
2 *Perform one pass over* (X, Y) *to partition it to the grid* G
3 **for each** $c_{ij} \in G$ **do**
4 $sortCellByX(c_{ij})$
5 **for** $i = 0$ **to** $m_x - 1$ **do**
6 **for** $j = 1$ **to** $m_y - 1$ **do**
7 **for** $k = 0$ **to** $j - 1$ **do**
8 $(dis, tiX) \leftarrow (dis, tiX) +$ mergeSortSouth(c_{ij}, c_{ik})
9 **for each** $c \in grid$ **do**
10 $(dis, tiX, tiY, tiXY) \leftarrow (dis, tiX, tiY, tiXY) + knight(c)$
11 **for** $j = 0$ **to** $m_y - 1$ **do**
12 **for** $i = 1$ **to** $m_x - 1$ **do**
13 **for** $k = 0$ **to** $i - 1$ **do**
14 $(dis, tiY) \leftarrow$ mergeSortEast(c_{kj}, c_{ij})
15 $dis \leftarrow dis +$ discordantCells(G)
16 $con \leftarrow n(n-1)/2 - dis - tiX - tiY - tiXY$
17 **return** $(con - dis)/\sqrt{(con + dis + tiX) \cdot (con + dis + tiY)}$

Pairs of cells in the same row or column The mergeSortSouth module is used to find the discordant and tied pairs (p_l, p_m) of observations, such that $p_l \in c_i, p_m \in c_j$ and cell c_j is in the same

grid column and south of cell c_i. Before running this module, the contents of c_i and c_j are sorted by X. Note that, since a pair ($p_l \in c_{ij}, p_m \in c_{ik}$) of observations are in different rows, there cannot be Y or XY ties. Our goal is to count the discordant pairs and those tied on X. We achieve this by merge-joining the X-sorted c_{ij} and c_{ik} cells. Algorithm 3 describes this process. We initialize two pointers (indices) $cur1$ and $cur2$ that run along the positions of the two arrays, progressing at each time the position with the smallest X value. If the observations at $cur1$ and $cur2$ have the same X-value (i.e., $x_i = x_j$), then we move forward both pointers to collect all observations with value equal to x_i and update the number tiX of tied X pairs, accordingly. We also update the number dis of discordant pairs to include the discordant observations in c_j with the points in c_i having the x_i value. If $x_i < x_j$, we update the discordant pairs, whereas if $x_j < x_i$ we do nothing.

Figure 3 illustrates the steps of Algorithm 3 and the contribution of each case to discordant and tied pairs. The first observation in X-order belongs to c_i, so we increase the discordant counter dis by $|c_j|$, as $cur2 = 0$; i.e., the first point of c_i is discordant with all points in c_j (Line 16). The second observation in X-order belongs to c_j, so we simply increment $cur2$ by 1 (Line 18). For the third X-value, we have $x_i = x_j$; we count 2 objects in c_i and 3 objects in c_j as X-tied, with $x_i = x_j$, so we increase tiX by $2 \times 3 = 6$ pairs (Line 11). At the same time, we increase the number of discordant pairs by 2×1 as one point in c_j has X-value larger than x_i (Line 12).



(a) 1st X value **(b) 2nd X value** **(c) 3rd X value**

Figure 3: Steps of mergeSortSouth.

The mergeSortEast module is symmetric to mergeSortSouth and applies on two cells c_i and c_j, such that c_i and c_j are in the same row and c_j is to the east of c_j. The difference is that the input cells are assumed to be sorted by Y and the comparisons in the merge process are done in the Y dimension to discover discordant and Y-tied pairs. As already explained, the application of Knight's algorithm in Line 9 of Algorithm 2 to each cell, changes the X-sort order in each cell to Y-sort order, so cells need not be sorted by Y before running mergeSortEast in Lines 11-14 of Algorithm 2. The pseudocode of mergeSortEast is omitted for brevity.

Discordant cells To complete the total number dis of discordant pairs, Line 15 of Algorithm 2 runs function discordantCells, which, for each cell c_{ij} multiplies the number $|c_{ij}|$ of points in c_{ij} with the total number of points in cells *south-east* of c_{ij}.

Algorithm 3: Merge sort join for south cells.

Input: Array of elements of cell (c_i) and array of elements of cell (c_j) that is located south of c_i, both arrays ordered by X
Output: A tuple with the discordant and X-tied pairs

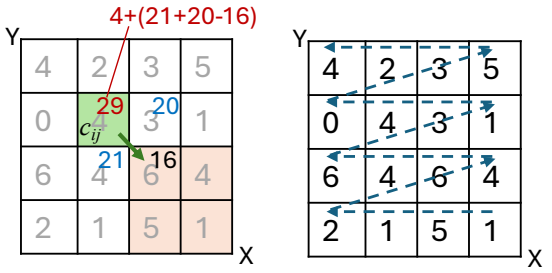
```
1  Function mergeSortSouth(c_i, c_j):
2      dis ← 0; tiX ← 0; cur1 ← 0; cur2 ← 0
3      while cur1 < |c_i| and cur2 < |c_j| do
4          x_i ← c_i[cur1].x; x_j ← c_j[cur2].x
5          if x_i = x_j then
6              iStart ← cur1; jStart ← cur2
7              while cur1 ≤ |c_i| and c_i[cur1].x = x_i do
8                  cur1 ← cur1 + 1
9              while cur2 ≤ |c_j| and c_j[cur2].x = x_j do
10                 cur2 ← cur2 + 1
11             tiX ← tiX + (cur1 − iStart) · (cur2 − jStart)
12             dis ← dis + (|c_j| − cur2) · (cur1 − iStart);
13             cur2 ← jStart
14         else if x_i < x_j then
15             cur1 ← cur1 + 1
16             dis ← dis + (|c_j| − cur2)
17         else
18             cur2 ← cur2 + 1
19     return (dis, tiX)
```

Algorithm 4 shows the details of discordantCells, which achieves this with a single pass over the grid. Specifically, starting from the southeast corner of the grid, in each cell c_{ij} we accumulate the total number of points in c_{ij} and all cells south, east, and southest of c_{ij}. This is done by adding the number of points $|c_{ij}|$ to $(A_{i,(j-1)} + A_{(i+1),j} - A_{(i+1),(j-1)})$, where $A_{i,j}$ denotes the total number of points in the cells at grid columns larger than or equal to i and grid rows smaller than or equal to j. For example, as Figure 4 illustrates, to compute $A_{i,j}$ for cell c_{ij} and all cells to the south, east, and southeast of c_{ij}, we can add to $|c_{ij}| = 4$ the corresponding sums $A_{i,(j-1)} = 21$ and $A_{(i+1),j} = 20$, to the south and east of c_{ij} and subtract the sum $A_{(i+1),(j-1)} = 16$ to the southeast of c_{ij} (since $A_{(i+1),(j-1)}$ is included in both $A_{i,(j-1)}$ and $A_{(i+1),j}$). After computing $A_{i,j}$, for the current cell c_{ij}, we contribute to the total number of discordant pairs dis the product $|c_{ij}| \cdot A_{(i+1),(j-1)}$, i.e., the number of points in c_{ij} times the total number of points aggregated at the southeast cell of c_{ij}. In Figure 4a, the contribution of c_{ij} to dis is $4 \cdot 16 = 64$. Note that Algorithm 4 does not keep the entire A matrix, but only updates the current row and uses the aggregates at the current and previous row. The order by which the matrix with cell statistics is processed is illustrated in Figure 4b.

3.1 Adaptive Grid Partitioning

The performance of intra and inter cell processing depends on the number of points in the processed cells. By employing a *regular* (i.e., uniform) grid, if the data are skewed, there is a risk of having a few heavy-loaded cells, that may render the partitioning process ineffective. To alleviate data skew, we adopt an *adaptive* grid partitioning to the data distribution. Specifically, we use a data sample to determine approximate quantiles at each dimension and use them to define the grid boundaries.

Figure 5 illustrates the process of defining adaptive grid boundaries, assuming that we need 3 partitions per dimension. Given a sample of observations, we order it by Y and take the 33% and



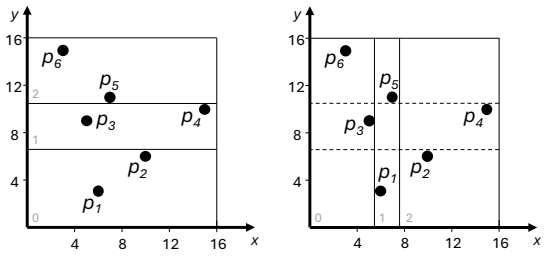
(a) Aggregate counts (b) Processing order of cells

Figure 4: Computing discordant pairs for south-east cells and updating aggregates.

Algorithm 4: Finding the discordant pairs between cells.

Input: A $m_x \times m_y$ grid G, where for each cell $c_{ij} \in G$, $G.|c_{ij}|$ is the total number of points in c_{ij}
Output: Discordant pairs

```
1  Function discordantCells(G, m_x, m_y):
2      dis ← 0; arr ← new array[m_x]
3      arr[m_x − 1] ← G.|c_{m_x−1,0}|
4      for i = m_x − 2 to 0 do
5          arr[i] ← G.|c_{i0}| + arr[i + 1]
6      for j = 1 to m_y − 1 do
7          diagonal ← arr[m_x − 1]
8          arr[m_x − 1] ← arr[m_x − 1] + G.|c_{m_x−1,j}|
9          for i = m_x − 2 to 0 do
10             below ← arr[i]
11             arr[i] ← arr[i] + arr[i + 1] − diagonal + G.|c_ij|
12             dis ← dis + G.|c_ij| · diagonal
13             diagonal ← below
14     return dis
```



(a) Partitioning across Y (b) Partitioning across X

Figure 5: Adaptive grid partitioning.

66% percentile Y-values to define the cell boundaries for Y. This guarantees (depending on the sample effectiveness) that each horizontal stripe gets an equal number of points. Then, we can repeat by sorting the sample by X to obtain equi-depth vertical stripes. Although adaptive grid partitioning does not guarantee balance at the cell level, it avoids hugely overloaded cells (which was a practical problem for regular grids), as in the worst case a cell does not take more than the data in the row or column where it belongs. Hence, in a $m \times m$ grid, no cell takes more than $1/m$ of the data.

3.2 Complexity Analysis

In this section, we analyze the computational cost of Algorithm 2. We assume that a $m \times m$ *adaptive* grid is used for the partitioning. Under this, the cost of assigning the observations to grid cells (and counting the number of points per cell) is $O(n \log m)$, as we need $O(\log m)$ time to determine the grid row and column for each observation, by binary search. The cost of X-sorting all cells (Lines 3–4 of Algorithm 2) is bounded by the worst-case distribution, where in each row (or column) of the grid, just one cell gets n/m points and the other cells are empty. This means that total cost of X-sorting is $O(m \cdot (n/m) \log(n/m))$, i.e., $O(n \log(n/m))$. When merging cells in the same column (Function mergeSortSouth), each observation is accessed $m-1$ times, as the cell that contains it is merged with $m-1$ other cells. Therefore, the total cost of all mergeSortSouth calls (Lines 5–8 of Algorithm 2) is $O(mn)$. The cost of applying Knight's algorithm for the intra-cell relationships (Lines 9-10 of Algorithm 2) is the same as the cost of Y-sorting each cell, i.e., the total cost for all cells is $O(n \log(n/m))$. The cost of merging east cells (Lines 11-14 of Algorithm 2) is the same as the total cost of mergeSortSouth calls, i.e., $O(mn)$ in total. Finally, discordantCells applies a linear pass over the cells with total cost $O(m^2)$.

Hence, the overall time complexity is $O(n(m+\log(n/m)))$. Based on this, m should be selected to a value close to $\log_2 n$; for values of m much larger than $\log_2 n$, $O(nm)$ becomes the dominant factor in the complexity and Algorithm 2 becomes more expensive than Knight's algorithm. More importantly, our algorithm offers opportunities for parallelism, as mergeSortSouth calls can be executed in parallel and independently by different threads. The same holds for all calls to Knight's algorithm for intra-cell pairs, and for all mergeSortEast calls. In Section 4, we will present a distributed and parallel version of our algorithm that greatly reduces the cost of Kendall's τ_b computation. The space complexity is $O(n)$, even for large grids, as we do not have to allocate memory for empty cells (non-empty cells can be accessed with the help of a hashmap).

4 DISTRIBUTED τ_b COMPUTATION

If the observations do not fit in the memory of a single machine, to compute the exact τ_b rank correlation, we need to resort to a distributed algorithm. In this section, we present such an approach that relies on data partitioning to the main memories of multiple nodes and applies MapReduce primitives to compute and aggregate the relationships between pairs required for τ_b in parallel.[1]

The data are partitioned and processed independently in different nodes, where each node provides the necessary information to a designated *coordinator* node. After collecting all the necessary information, the coordinator node calculates the coefficient. Figure 6 illustrates the distributed algorithm and Algorithm 5 presents its details. The coordinator node defines the *adaptive* grid boundaries based on a sample of the data and broadcasts them to the nodes. Then, the data are shuffled and partitioned to horizontal stripes based on the horizontal lines of the grid that divide the Y axis. All

[1]Implementing Knight's algorithm [19] on Spark is challenging, because there is no efficient way of merging sorted runs that appear in different nodes. Multi-way merging requires a lot of communication and binary merging requires multiple rounds of merging with data transfers (partitioning/shuffling multiple times).

data in the same horizontal stripe end up at the same worker node. For each horizontal stripe, Algorithm 5 runs Knight's algorithm to compute all intra-stripe discordant and tied pairs independently at each worker. After processing a stripe, the corresponding worker node dispatches the number of discordant, X-tied, Y-tied, and XY-tied pairs to the coordinator. The coordinator aggregates these statistics and maintains the global values of these variables.

After finishing with all horizontal stripes, the data are reshuffled and partitioned again based on the grid divisions across the X axis, such that all data of a single vertical stripe end up at the same node. The nodes then process the vertical stripes independently. Since intra-cell pairs have already been considered by horizontal stripes, vertical stripes are processed differently (Function southMSReducer). Specifically, each node divides the vertical stripes into cells (using the grid boundaries), counts the number of elements $|c_j|$ per cell c_j and sorts the cell by X. Then, each cell c_j in the vertical stripe is merge-joined with all cells c_k at c_j's south using mergeSortSouth (Algorithm 3) to find the inter-cell discordant and X-tied pairs in the same grid column. All counters (i.e., dis, tiX) are collected by the coordinator and aggregated. The coordinator also collects, from the workers that execute southMSReducer, a 2D histogram the number of points in each cell. The coordinator uses the collected cell counts to run discordantCells module (Algorithm 4) that measures the discordant pairs in different rows and columns of the grid and completes the computation of τ_b.

Load balancing and complexity. The Reduce jobs of the horizontal (and vertical) stripes have equal complexity, ensured by the adaptive grid defined by the coordinator (Line 2 of Algorithm 5). This leads to load balancing and good scalability with increasing numbers of workers, as we demonstrate experimentally. The processing of horizontal stripes (Function knightReducer) is the computational bottleneck of the algorithm. Assuming a $m \times m$ grid, each of the m stripe includes n/m points and requires $(n/m)\log(n/m)$ time to process. Hence, if k jobs can be executed in parallel, the time complexity is bounded by $O(\frac{1}{k} \cdot m \cdot (n/m)\log(n/m))$, which is $O(\frac{n}{k}\log(\frac{n}{m}))$.

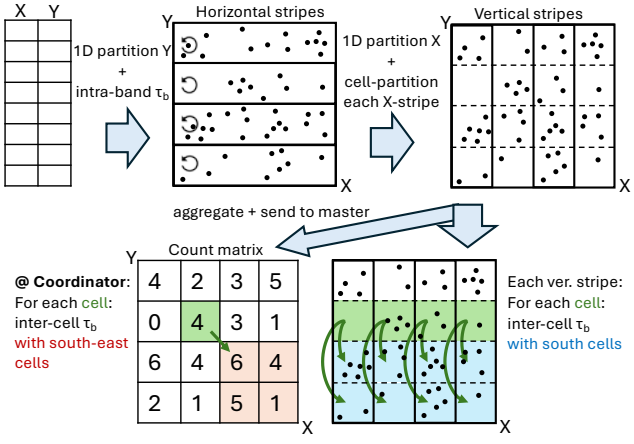


Figure 6: Overview of distributed Kendall τ_b computation.

Algorithm 5: Distributed Grid Algorithm.

Input: A $m_x \times m_y$ grid G, a 2D array P of observations
Output: $\tau_b(P)$

1 $dis \leftarrow 0$; $tiX \leftarrow 0$; $tiY \leftarrow 0$; $tiXY \leftarrow 0$
2 *Based on a sample from P, define an adaptive grid G*
3 $P' \leftarrow P$.mapToYStripe.$groupByKey()$
4 $(dis, tiX, tiY, tiXY) \leftarrow P'$.knightReducer$(key, points)$.$sumResults()$
5 $P'' \leftarrow P$.mapToXStripe.$groupByKey()$
6 $(dis, tiX) \leftarrow$
  $(dis, tiX) + P''$.$southMSReducer(key, points)$.$sumResults()$
7 $dis \leftarrow dis$+discordantCells(G, m_x, m_y)
8 $con \leftarrow |P|(|P|-1)/2 - dis - tiX - tiY - tiXY$
9 **return** $(con - dis)/\sqrt{(con + dis + tiX) * (con + dis + tiY)}$
10 **Function** mapToYStripe$(pointsChunk)$:
11 $mappedPoints = []$
12 **for each** $p \in pointsChunk$ **do**
13 $stripeId \leftarrow (grid.stripeY(dimension, p))$
14 $mappedPoints.append((stripeId, p))$
15 **return** $mappedPoints$
16 **Function** mapToXStripe$(pointsChunk)$:
17 *Same as* mapToYStripe(), *but assigns points to X stripes*
18 **Function** knightReducer$(key, points)$:
19 $sortedPoints \leftarrow sortByX(points)$
20 $(dis, tiX, tiY, tiXY) \leftarrow knight(sortedPoints)$
21 **return** $(dis, tiX, tiY, tiXY)$
22 **Function** southMSReducer$(key, points)$:
23 $cells \leftarrow assignPoints(points)$; $dis \leftarrow 0$; $tiX \leftarrow 0$
24 **for each** $c_j \in cells$ **do**
25 $sortCellByX(c_j)$
26 **for** $j = 1$ **to** $m_y - 1$ **do**
27 **for** $k = j - 1$ **to** 0 **do**
28 $(dis, tiX) \leftarrow (dis, tiX)$+mergeSortSouth$(c_j, c_k)$
29 **return** (dis, tiX)

5 APPROXIMATE COMPUTATION OF τ_b

We now present an approximate algorithm of Kendall's τ_b, which avoids the expensive steps of intra-cell comparisons and inter-cell merge-scans for pairs of cells at the same grid row or column.

Algorithm 6 shows the steps of our approximate algorithm. As in the exact algorithm, we define an adaptive grid, but do not partition the data into the grid's cells; instead, we only measure the number of points $|c|$ that fall in each cell c. Then, we run an adaptation of discordantCells algorithm (see Algorithm 4), where for each cell c_{ij}, we (i) compute $A_{i,j}$, the total number of points in c_{ij} and all cells south, east, and southeast of c_{ij}; and (ii) we update dis, by multiplying $|c_{ij}|$ with the sum of counts $A_{i+1,j-1}$ recorded in the neighbor cell diagonally to the south-east.

To complete the contribution of c_{ij} we have to take into account the contribution to τ_b from intra-cell pairs in c_{ij} and inter-cell pairs from c_{ij} and cells to the south of c_{ij} and to the east of c_{ij}. We employ estimates of these contributions, as we aim for a fast approximate algorithm (otherwise, we would run the steps of our exact method). To simplify processing, we ignore ties, as estimating ties requires advanced statistics (histograms of distinct X and Y values per cell). We assume that the points in each cell are *uniformly distributed* and compute the expected number of discordant pairs for each cell or pair of cells that we consider. In particular, the intra-cell discordant pairs in c_{ij} are estimated to be half of the pairs in c_{ij}, i.e., $|c_{ij}| \cdot (|c_{ij}| - 1)/4$. The inter-cell discordant pairs in two cells

c_i and c_j, where either c_j is the same column as c_i and to the south or c_j is the same row as c_i and to the east are estimated to be half of the pairs, i.e., $|c_i| \cdot |c_j|/2$.

Overall, Algorithm 6 scans the grid just once, row-by-row from east to west and from south to north, in the same manner as Algorithm 4. At each cell c, the approximate algorithm (i) estimates the intra-cell discordant pairs (Lines 4, 7, 12, and 18), (ii) estimates the inter-cell discordant pairs in c and in cells to the east of c (Lines 8 and 19), (iii) estimates the inter-cell discordant pairs in c and in cells to the south of c (Lines 13 and 20), and (iv) computes exactly the discordant pairs in c and in all cells south-east from c (Line 17). The concordant pairs are computed as a complement of the discordant ones (Line 22) and the tied pairs are assumed to be 0.

Complexity. Algorithm 6, in Line 1, reads all observations and identifies the cell wherein each observation falls. This takes $O(n \log m)$ time, assuming a $m \times m$ grid. Thereafter, it performs a linear scan over all cells and updates the *dis* statistics. A constant number of operations are executed at each cell, so the total cost of the algorithm is $O(n \log m + m^2)$, or simply $O(n \log m)$, assuming that $m^2 < n$. This is much lower than the $O(n(m + \log(n/m)))$ time required by our exact algorithm and Knight's $O(n \log n)$ complexity. More importantly, our approximate algorithm is lightweight as it requires $O(m^2)$ space vs. $O(n)$ of the exact algorithm.

Approximation error bound. Consider $m \times m$ *adaptive* grid, where each row and column of the grid takes $x = n/m$ observations. The worst-case error is when, in each row and column, all observations fall in the same cell and in all such cells the points are perfectly correlated or in all such cells the points are perfectly anti-correlated. Since the algorithm incurs no errors in the inter-cell discordant pairs (no pair of cells are on the same row/column) the only errors are for the discordant pairs which are estimated to be $x(x-1)/4$ instead of the actual 0 or $x(x-1)/2$, where $x = n/m$. Hence, the maximum error in the discordants count in each non-empty cell is $x(x-1)/4$ and in all cells is $mx(x-1)/4$. Summing up, the maximum absolute error is $\frac{mx(x-1)/4}{n(n-1)/2}$, which is equal to $\frac{n-m}{2m(n-1)}$. For $n >> m$, the bound is $1/2m$. For example, when using a 50×50 adaptive grid, the absolute error in the correlation computation is 0.01 or lower.

5.1 Advanced Approximate Algorithm

Our algorithm relies on uniformity assumption at each cell and ignores possible ties, so it may reach its upper error bound when the data distribution in cells is non-uniform. To alleviate this issue, we suggest a more sophisticated version of Algorithm 6, which employs a small data sample and histograms at each cell to improve its accuracy, especially in skewed datasets that include many ties.

In particular, while scanning the data to determine the number of points $|c_{ij}|$ in each cell c_{ij}, we apply independent sampling, to keep a small sample s_{ij} of the points in c_{ij}. For each point that we read from the input, we keep it as a sample with a small probability (empirically set to 0.01). We also divide each cell c_{ij} into k horizontal and into k vertical stripes uniformly. While reading the data, we keep $2k$ values in the cell which form two k-equiwidth histograms that capture the X- and Y- distribution of values in c_{ij}. Finally, for each of the k horizontal and vertical stripes in the cell, we keep

Algorithm 6: Approximate Grid-based Algorithm.

Input: A $m_x \times m_y$ grid G, a 2D array (X, Y) of n observations
Output: Approximate $\tau_b(X, Y)$

1 *Perform one pass over (X, Y) to count for each cell $c \in G$, the number $|c|$ of observations in c.*
2 $arr \leftarrow new\ array[m_x]$; $dis \leftarrow 0$
3 $arr[m_x - 1] \leftarrow G.|c_{m_x-1,0}|$
4 $dis \leftarrow G.|c_{m_x-1,0}| \cdot (G.|c_{m_x-1,0}| - 1)/4$ `// intra-cell`
5 **for** $i = m_x - 2$ **to** 0 **do** `// scan cells in row 0 (bottom)`
6 $arr[i] \leftarrow G.|c_{i,0}| + arr[i+1]$
7 $dis \leftarrow dis + G.|c_{i,0}| \cdot (G.|c_{i,0}| - 1)/4$ `// intra-cell`
8 $dis \leftarrow dis + G.|c_{i,0}| \cdot arr[i+1]/2$ `// cells to the east`
9 **for** $j = 1$ **to** $m_y - 1$ **do** `// row j`
10 $diagonal \leftarrow arr[m_x - 1]$
11 $arr[m_x - 1] \leftarrow arr[m_x - 1] + G.|c_{m_x-1,j}|$
12 $dis \leftarrow dis + G.|c_{m_x-1,j}| \cdot (G.|c_{m_x-1,j}| - 1)/4$ `// intra-cell`
13 $dis \leftarrow dis + G.|c_{m_x-1,j}| \cdot diagonal/2$ `// cells to the south`
14 **for** $i = m_x - 2$ **to** 0 **do** `// scan cells in row j`
15 $below \leftarrow arr[i]$
16 $arr[i] \leftarrow arr[i] + arr[i+1] - diagonal + G.|c_{ij}|$
17 $dis \leftarrow dis + G.|c_{ij}| \cdot diagonal$ `// south-east cells`
18 $dis \leftarrow dis + G.|c_{i,j}| \cdot (G.|c_{i,j}| - 1)/4$ `// inter-cell`
19 $dis \leftarrow dis + G.|c_{i,j}| \cdot (arr[i+1] - diagonal)/2$ `// east`
20 $dis \leftarrow dis + G.|c_{i,j}| \cdot (below - diagonal)/2$ `// south`
21 $diagonal \leftarrow below$
22 $con \leftarrow n \cdot (n-1)/2 - dis$ `// estimate of concordants`
23 **return** $(con - dis)/(con + dis)$

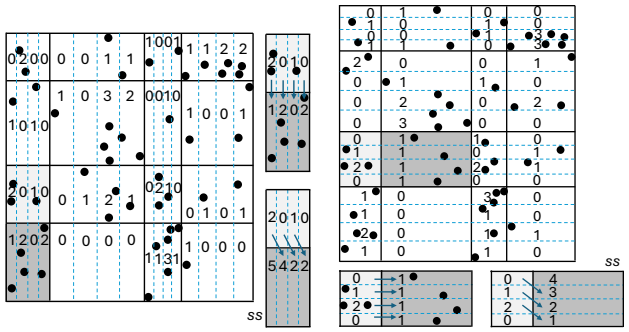
the points in the sample s_{ij} that fall in the stripe. We now apply Algorithm 6 with the following changes.

5.1.1 Intra-cell statistics. For each cell c_{ij}, we use the sample s_{ij} to compute all statistics (discordant pairs, tied X pairs, etc.). Each statistic is multiplied by the ratio (pairs in c_{ij})/(pairs in s_{ij}) to estimate the corresponding statistics in c_{ij}. Note that if s_{ij} is empty, but $|c_{ij}| > 0$, we apply the uniform distribution assumption as in the original algorithm.

5.1.2 Inter-cell statistics. Assume that c_p is north of c_q (the case where c_p is west of c_q is symmetric). For each position i of the X-histograms of c_p and c_q, let $|c_p^i|$ and $|c_q^i|$ be the actual number of points there and s_p^i and s_q^i be the sample point-sets kept there. s_p^i and s_q^i are used to measure discordant and X-tieX statistics, which are multiplied by $(|c_p^i| \cdot |c_q^i|)/(|s_p^i| \cdot |s_q^i|)$ to derive the estimations that will be aggregated into the global discordant and X-tied statistics for cells c_p and c_q. In addition, for each position i of the X-histograms, we multiply $|c_p^i|$ with all $|c_q^j|$, such that $i < j$ and add the products to the global number of discordant pairs. To accelerate the last part, we compute a *suffix-sum* (ss) histogram of each X-histogram, where each entry $|ss_p^i|$ of the suffix histogram at cell c_p holds $\sum_j |c_p^j|, \forall j \geq i$, and multiply each $|c_p^i|$ with $|ss_q^{i+1}|$.

Example Figure 7 shows an example dataset with an adaptive grid (solid lines), where each row and column takes 12 points. In Figure 7a, each cell c_p of the grid is split vertically to $k = 4$ partitions to define a X-histogram. The numbers at the columns in a cell show the X-histogram values. For each histogram position, we also keep a sample (not shown in the figure). To estimate the τ_b statistics for all pairs of points in two cells c_p and c_q in the same grid column (e.g., c_p is the light-shaded cell and c_q the dark-shaded one), we first use the samples at each position i of the X-histograms to estimate the

statistics for all pairs of points in the corresponding bands of c_p and c_q. Second, for each position i of c_p's X-histogram we multiply the number of points $|c_p^i|$ in it with the number ss_q^{i+1} at the next position $i+1$ in the suffix-sum X-histogram of cell c_q (see the dark-shaded cell at the bottom-right corner of Figure 7a), and sum up these to derive the exact number of discordant pairs. Figure 7b illustrates a symmetric procedure that is applied using the Y-histograms.



(a) X-histograms and computation of correlation statistics for pairs of cells in the same column

(b) Y-histograms and computation of correlation statistics for pairs of cells in the same row.

Figure 7: Advanced approximate algorithm.

6 EXPERIMENTAL EVALUATION

In this section, we evaluate our approach, proposed in Section 3 against Knight's algorithm (Apache Commons' implementation). We also evaluate the efficiency and scalability of the distributed version of our algorithm, presented in Section 4. Finally, we experimentally study the accuracy and cost of our approximate algorithm for Kendall's τ_b (Section 5). Our experimental study aims to answer the following research questions. **RQ1:** Under what conditions does our method outperform Apache Commons' implementation of Knight's algorithm [19]? **RQ2:** Does adaptive grid partitioning improve the efficiency of Algorithm 2, compared to regular grid partitioning? **RQ3:** What is a good grid resolution for the centralized and distributed versions of our method? **RQ4:** Does our distributed algorithm scale well when increasing the input size and the number of executors? **RQ5:** How efficient and effective is our approximate Kendall τ_b computation method?

6.1 Experimental Setup

Datasets. We use six synthetic datasets, each with a different distribution in the 2D space, and four real ones from diverse domains. The datasets are visualized in Figure 8. By default, we used a sample of 80M points from each dataset as our input. Most synthetic datasets were generated using the Spider data generator [16] and their names are self-descriptive. We also employ two highly skewed synthetic datasets: Hotspots and Zipfian. Hotspots groups the vast majority of data in two clusters with the same (very small) standard deviation, each having 60% and 35% of all data points, respectively; the remaining 5% of the points are uniformly distributed. Zipfian was produced by mixing two Zipf independent distributions with $\alpha = 1$. The real datasets cover various domains, namely

Parameter	Cardinality
Dataset size	15M–**80M** (small), **250M**–1B (large)
Grid resolution	10x10, 25x25, 50x50, 100x100, 200x200, 400x400
Small-scale dataset size	15M, 30M, 45M, 60M **(25x25)**
Large-scale dataset size	250M, 500M, 750M, 1B **(200x200)**
Executors	3, 6, 9, **12 (250M)**

Table 1: Overview of the datasets used in the experiments.

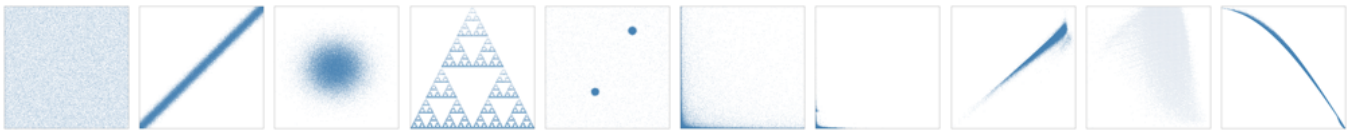
internet speed, astronomy, weather data, and solar radiation. The internet speed dataset [9] contains speed test metrics from Ookla (www.speedtest.net), where from we select download speed and latency to measure correlation. The Gaia DR3 dataset [4] contains records of celestial objects; we select the mean magnitude in the G-band (white light, broad bandpass) and the mean magnitude in the RP-band (red photometer) to measure correlation. The Weather dataset includes climatological data from NCEI [12]; we measure the τ_b correlation between bulb temperature and relative humidity. Finally, the Radiation dataset [36] is provided by the National Solar Radiation Database and contains solar radiation metrics across the United States; here, we measure how solar zenith and extraterrestrial radiation per sq. meter are correlated.

Platform. For the experiments, we employed a cluster of 15 virtual machines; all VMs were equipped with 4 CPU cores, 8GB RAM and 30GB system disk, running Ubuntu 16.04.6 LTS. Twelve of these VMs had a mounted disk of 102GB, running HDFS v3.2.1 and functioning as Datanodes and NodeManagers. The attachable disks of the cloud platform are based on the Ceph storage system (ceph.io), which is a distributed block level storage. The remaining 3 VMs acted as NameNode, ResourceManager (YARN) and Driver (i.e., Coordinator) for the running jobs. The HDFS was set to the default configuration, i.e. 128MB block size and replication factor of 3. For the experiments with the local executions, the VM that was used was the one that acted as a Driver for the conduction of experiments in the distributed mode. The IaaS platform we use for the experiments uses under-the-hood SSDs (for persistent storage) on which Ceph runs as a distributed storage solution for supporting the whole infrastructure.

Metrics and parameters. In all experiments we measure the total wall clock time for a computation of Kendall's τ_b. For the local (centralized) experiments we set the maximum heap size of the Java Virtual Machine (JVM) to 4GB. To ensure consistent measurements, each job runs to completion, after which the JVM is terminated. Thus, every job starts on a fresh JVM instance. Concerning the distributed experiments, we set the number of executors in Spark equal to the number of used processing nodes (12), each one having 4GB main memory and 2 cores per executor. We also set the driver's main memory to 5GB and the number of Spark partitions for the data shuffling (*groupByKey* operation) to 192. In both local and distributed experiments, each reported measurement is obtained by averaging the execution time of five runs.

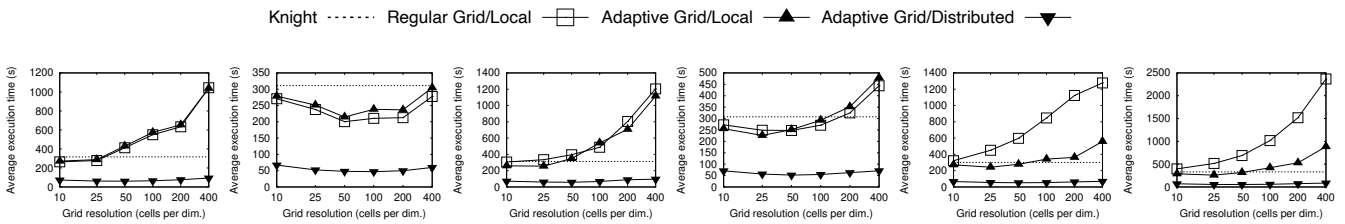
6.2 Experimental Results

Table 1 shows the ranges of parameter values used in the experimental instances (default values in bold), for the number n of



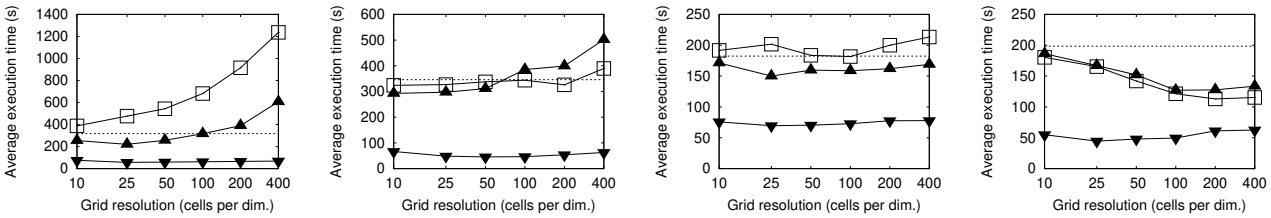
(a) Uniform	(b) Diagonal	(c) Gaussian	(d) Sierpinski	(e) Hotspots	(f) Zipfian	(g) Ookla	(h) Gaia	(i) Weather	(j) Radiation
$\tau_b = 0$	$\tau_b = 0.94$	$\tau_b = 0$	$\tau_b = 0$	$\tau_b = 0.44$	$\tau_b = 0$	$\tau_b = -0.34$	$\tau_b = 0.83$	$\tau_b = -0.21$	$\tau_b = -0.98$

Figure 8: Visualization of the synthetic and real datasets. From left to right: synthetic (first six) and real (last four).



(a) Uniform (b) Diagonal (c) Gaussian (d) Sierpinski (e) Hotspots (f) Zipfian

Figure 9: Average execution time of Kendall τ_b computation (synthetic datasets, n=80M).



(a) Ookla (b) Gaia (c) Weather (d) Radiation

Figure 10: Average execution time of Kendall τ_b computation (real datasets, n=80M).

observations (i.e., dataset size), grid resolution, number of executors in distributed processing. To determine the adaptive grid, we used a sample size equal to the square root of the dataset size; we have experimentally found that this leads to a balanced partitioning in rows and columns.

6.2.1 Grid type and resolution. The first set of experiments targets RQ1–RQ3. We compare our method with the Apache Commons implementation of Knight's algorithm on a single machine (local) for various grid resolutions and n=80M input sizes. The study includes our grid-based algorithm (Algorithm 2) run locally on a single machine and the distributed version of our method (Algorithm 5). We tested each version equipped with a regular (i.e., uniform) grid with equi-width partitions and our suggested adaptive grid (Section 3.1). We present the effect of varying the grid resolution on the synthetic and real datasets in Figures 9 and 10, respectively. In all plots, the horizontal axis represents the number of partitions m per dimension (i.e., the grid is $m \times m$), while the vertical axis stands for the execution time in seconds (an average over 5 runs). The dotted horizontal line in each plot shows Knight's algorithm performance, which does not use a grid.

Due to the data skewness, in most synthetic datasets and in all real datasets, a regular grid could not be used within our distributed algorithm. In these cases, few regular stripes become overloaded with data, thus exhausting the memory of the executors that host them, while the rest of the executors remain underutilized or idle. On the other hand, there is no issue in using our distributed algorithm with an adaptive grid in all cases, since load is equally balanced to stripes, as explained in Section 4. Hence, we only present results of our distributed algorithm using an adaptive grid.

From the experimental results of Figures 9 and 10, we draw the following conclusions:

- As expected, in (relatively) uniform datasets, there is no difference between a regular and an adaptive grid in our centralized algorithm. On the other hand, when there is significant data skew (Hotspots, Zipfian, Ookla), using an adaptive grid brings a great benefit to the centralized algorithm. As, a result in the rest of the experiments, we use only adaptive grids in our methods.
- In almost all cases, for our centralized algorithm, the best grid resolution parameter m is close to $\log_2 n$, which is in accordance to our analysis in Section 3.2 (i.e., $n = 80M$, so $\log_2 n \approx 25$). For such values, our method outperforms Knight's. On the other hand, for large values of m, our centralized algorithm is outperformed by Knight's (see Section 3.2). Diagonal and Radiation are exceptional cases, where our method widely outperforms

Knight's and its performance is best for relatively large values of m. In these two datasets, all points are within a narrow band close to a diagonal, and the rest of the space is empty; hence a larger grid resolution results in numerous empty cells, and the calls to the time-consuming functions (intra-cell Knight's, mergeSortSouth, and mergeSortEast) are minimized.
- Our distributed algorithm is robust with respect to the grid resolution parameter m and performs best for a wide range of m values. Its runtime is consistently several times lower than that of our centralized method and Knight's. Very small m values ($m < \log_2 n$, e.g., $m = 10$) are not good because this number is smaller than the resources (e.g., nodes) to be utilized and large values are also not good because the overall computational cost increases, as per our analysis at the end of Section 4. Once again, for our distributed algorithm the best value for the grid resolution parameter m is around $m = \log_2 n$.

6.2.2 Scalability to input data size. In the next set of experiments, we evaluate the scalability of the centralized and distributed algorithms with respect to the number of observations in the input dataset. Figure 11 uses both real and synthetic data of small scale that fit in the memory of a single VM, allowing the centralized algorithms (Knight's and our Algorithm 2 with both grid types) to run on a single host. We have fixed the grid resolution to $m = 25$ cells per dimension (25x25) and (i) generated (Gaussian and Serpinski or (ii) sampled (from Gaia and Weather) datasets of 15M, 30M, 45M, 60M instances. Observe that the gap between the relative performance of our centralized algorithm using an adaptive grid and Knight's algorithm improves as the scale of data increases, which is consistent with our complexity analysis (Sec. 3.2); for a fixed value of m, we expect relatively better performance for our method as the input size n increases.

Figure 12 evaluates the impact that the data scale has on the performance of our distributed algorithm equipped with an adaptive grid (RQ4). This time we used data of significantly larger scale (250M to 1B). As in the previous experiment, we generated synthetic data that follow Gaussian and Sierpinski distributions. From real datasets, we picked Gaia, the only dataset from those we have that scales beyond 1B. As the figure shows, the algorithm scales linearly to the input data size, regardless of the data distribution.

6.2.3 Scalability to number of executors. In Figure 13, we assess the impact of the number of executors in distributed execution on the performance of finding the τ_b score (RQ4). The horizontal axis represents the number of executors, while the vertical one shows average execution time in seconds. As expected, regardless the dataset used, increasing the number of executors leads to a proportional drop in execution time.

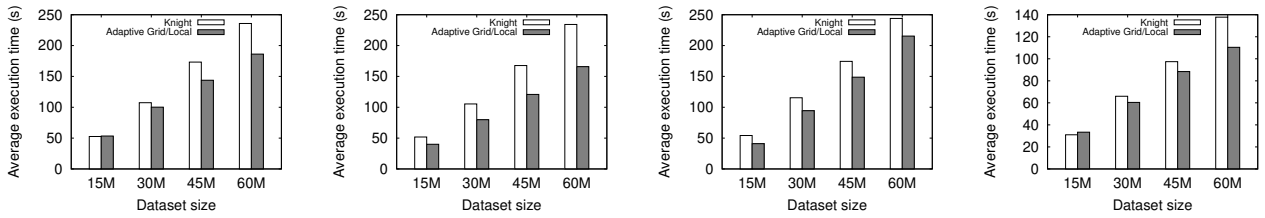
6.2.4 Load balancing. To achieve load balancing in distributed processing for computing Kendall's τ, we utilize the *LPT (Longest Processing Time)* heuristic for multiprocessor scheduling [13]. Using the data sample based on which the adaptive grid is defined, we assign stripes to workers based on the number of observations in each stripe, prioritizing the most populated stripes. This way, workers take either few long jobs or numerous small jobs and their load is balanced. To measure the effectiveness of LPT as a load balancing mechanism, we compute the *Maximum Relative Idle Time* (MRIT), defined as the difference between the fastest and the slowest executor's completion times over the total completion time. In an ideal scenario, all executors would complete simultaneously and MRIT is 0. For the Uniform, Diagonal, Gaussian, Sierpinski, Hotspots and Zipfian datasets, these delays are 0.06, 0.03, 0.12, 0.056, 0.084 and 0.11 respectively. For the Ookla, Gaia, Weather and Radiation datasets, MRIT is these delays are 0.07, 0.065, 0.077, and 0.04, respectively. This experiment shows that in all cases the idle time at each executor is up to a relatively small percentage of the total time and the load is well balanced.

6.2.5 Cost breakdown. To understand the impact of the different modules of our (adaptive) grid-based algorithms on the overall cost, in Figure 14 we show the cost of the different phases of our methods, from data loading and partitioning to computing and aggregating the rank correlation statistics. The labels of the phases on the stacked bars are self-explanatory based on the pseudocodes of the two algorithms. Note that there are no major bottlenecks in the processing of the cells. In all cases, the bottleneck is the data loading part; the east and south parts are reasonably similar (depending on the nature of the dataset), and the self-join within the cells (using Knight's algorithm) is reasonably small with the exception of Radiation, where, due to its special distribution (see Figure 8j), most computational work is done by Knight's calls.

6.2.6 Approximation accuracy and performance. In the last set of experiments, we evaluate the accuracy and cost of our approximate algorithm (RQ5), run locally on a single machine. Figure 15 shows the absolute error and execution time of Algorithm 6. As expected, higher grid resolutions lead to lower approximation errors and the time on top of the overhead to load the data grows linearly with m^2. Quite importantly, the absolute error soon drops below 10^{-2}. Datasets strongly correlated (e.g., Diagonal, Gaia) or anticorrelated (e.g., Radiation) have the largest errors, as the uniformity assumption in each cell fails. In Ookla and Gaia, for large m values, the error stabilizes instead of dropping due to many undetected ties (recall that Algorithm 6 assumes only discordant and concordant pairs). Still, for $m = 100$, the error is lower than 10^{-2} in all cases. The execution time remains an order of magnitude lower compared to the exact locally run algorithm. It spends about 30 seconds for data reading and counting cell statistics (Line 1 of Algorithm 6) and only needs several more seconds to produce the τ_b estimation, even when using large grids.

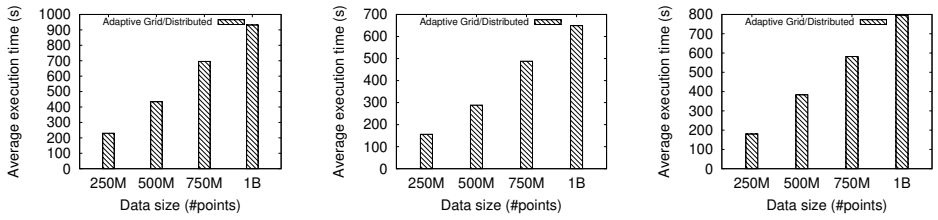
Figure 16 shows the performance of the advanced version of Algorithm 6, described in Section 5.1. Observe that using histograms and sampling leads to error drop by orders of magnitude, especially when the grid granularity is small. This is due to the detection of correlation and ties in cells and between them with the help of the samples and histograms. The advanced algorithm has increased cost compared to Algorithm 6, but the difference is insignificant when $m \leq 100$, especially considering the huge error drop it achieves.

6.2.7 Lessons learned. To sum up, our centralized algorithm performs better than the standard implementation of Knight's algorithm when it is equipped with a relatively coarse adaptive grid (e.g., $m = 25$) on large datasets. As expected, using an adaptive grid is more robust than using a regular grid, and much more efficient on skewed datasets (RQ1–RQ3). Second, employing a distributed



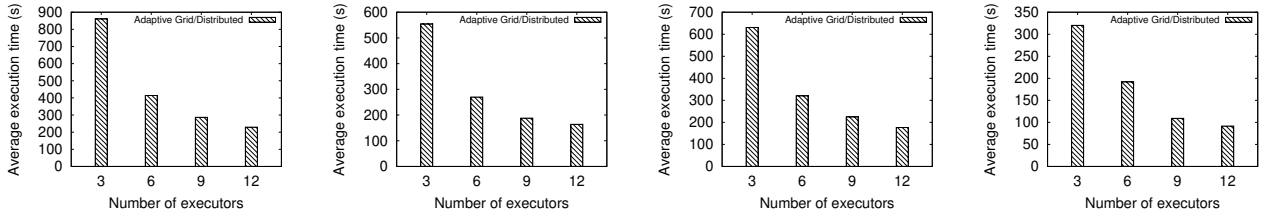
(a) Gaussian (b) Sierpinski (c) Gaia (d) Weather

Figure 11: Average execution time on the small-scale datasets (25x25 grid).



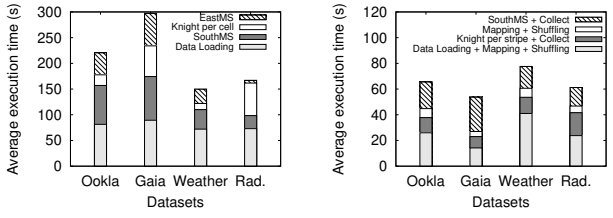
(a) Gaussian (b) Sierpinski (c) Gaia

Figure 12: Average execution time with varying data size on the large-scale datasets (200x200 grid).



(a) Gaussian (b) Sierpinski (c) Gaia (d) Weather

Figure 13: Average execution time with varying executors (200x200 grid).



(a) Centralized (25x25) (b) Distributed (200x200)

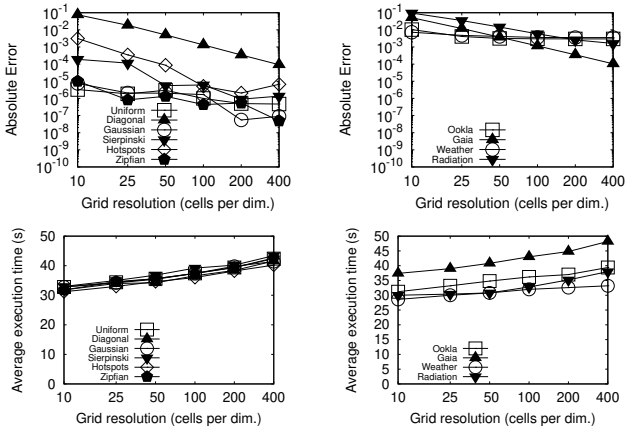
Figure 14: Cost breakdown of grid-based τ_b (real datasets).

system significantly accelerates computations compared to the local execution; our distributed algorithm scales gracefully with the data scale and the number of executors (RQ4). Overall, our distributed algorithm is the winner method and, more importantly, it exhibits remarkable robustness over a wide range of grid resolutions. Finally, our approximate algorithm is very accurate, especially for large grid resolutions (e.g., $m = 100$), while having one order of magnitude lower cost compared to the exact method (RQ5).

7 RELATED WORK

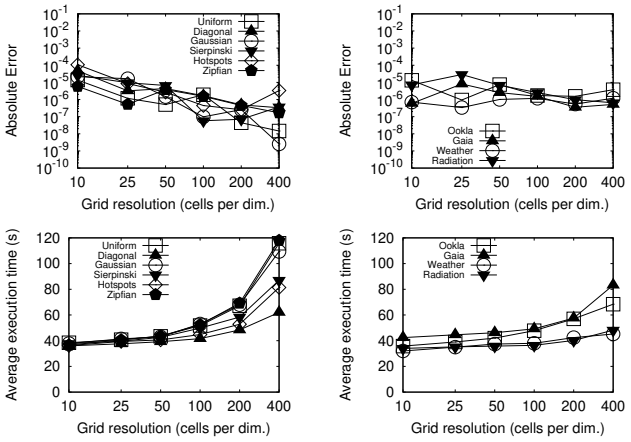
Applications of Kendall's τ. Arndt et. al [5] focus on the analysis of psychiatric symptom data, advocating that Kendall's τ coefficient is the better than Spearman's ρ from a statistical standpoint. Lapata [21] showed the reliability of Kendall's τ coefficient for automatic information ordering evaluation. Chen et al. [8] endorse Kendall's τ coefficient an estimator of critical transitions and regime shifts in complex systems. In the field of bioscience, τ has been applied to clinical survival results of patients with esophageal, pancreatic, and lung carcinomas [15]. In environmental science, Kendall's τ coefficient is applied on a study of oribatid mites in the peat blanket surrounding a bog lake, to detect correlated species groups [22]. In psychology, researchers use the family of Kendall's τ coefficients to assess trends and monotonicity [6]; the coefficients are used by psychologists as a vector of τ scores to measure how pre- and post-treatment phases differ, by calculating a τ-based trend.

Scalability in correlation computation. The large majority of related work in correlation computation from the area of data management focuses only on Pearson's coefficient and does not consider



(a) Synthetic (b) Real

Figure 15: Absolute error in τ_b score and average execution cost by Algorithm 6, varying grid resolution.



(a) Synthetic (b) Real

Figure 16: Advanced approximate algorithm using 1% sampling and 10-equiwidth X- and Y- histograms in each cell.

Kendall's τ at all. Chang et al. [7] and Eslami et al. [11] utilize GPU hardware for parallel processing to compute Pearson's correlation, while Mu et al. [26] compute Pearson's r in a cluster of nodes adopting the MapReduce paradigm. Zhang et al. [42] propose a Pearson correlation-based weighted optimization framework to define groups of variables that exhibit similar evolutionary trends. Xiong et al. [38] search for strongly correlated item pairs in market-basket databases, given a minimum correlation threshold [38]. To this end, the authors identify an upper bound of Pearson's correlation for binary variables which is cheap to compute and exhibits a special monotone property which allows pruning of many item pairs. Socha et al. [33], examine three algorithms for computing Pearson's coefficient for differential power analysis attacks. Spearman's rank correlation coefficient has been used on large-scale datasets in a parallel and distributed computing environment for enhancing recommendation systems [28]. Specifically, the coefficient is used to group data points, aiming at producing high quality correlation clusters. Recall, however, that Spearman's coefficient cannot handle ties effectively.

Due to the differences in the definitions and core computational methods of the three coefficients (Pearson's, Spearman's, Kendall's), it not possible to exploit previously proposed techniques for computing Pearson's and Spearman's coefficients to efficiently compute Kendall's τ: Pearson's r requires just two linear passes, whereas Spearman's ρ cannot handle ties. Mutual information (MI) is yet another correlation measure, studied by Ho et al. [14] for pairs of time series, restricted to a time window and considering delays in the time-series events. To the best of our knowledge, due to the essential inherent complexity of Kendall's coefficient, the need to handle ties, and the existence of lower-complexity algorithms like Pearson's, the state of the art does not yet include a scalable, paralellizable, tie-aware method to compute Kendall's correlation.

Other work. Our work also relates to studies of computational problems in which space partitioning is applied to split the workload in smaller parts and process them in parallel. These studies focus on parallel skyline operation [27, 35], spatial joins [20, 30, 34] and joins that include spatial and textual predicates [10, 43]. These techniques are not applicable to the computation of τ_b, as the problem definitions and objectives are different. For example, although there is work on grid-based and quadtree-based partitioning for parallel and distributed computation of skylines (e.g., [2, 29]), our problem is confined to 2D only and to computing a single value (as opposed to a subset of points). In addition, our problem requires, for each cell and stripe, to access and compare all points in it. On the other hand, when using a grid (or any other partitioning method) for skyline computation, numerous cells (or partitions) can be pruned by a single point (if they are dominated by it) and do not have to be processed. In spatial joins, only points in the same cell or partition need to be compared to each other and remote pairs of points are pruned; the main goal is to achieve this property while minimizing data replication in the partitioning phase [20].

8 CONCLUSIONS

We proposed a scalable and efficient geometric approach for the computation of Kendall's τ_b coefficient for very large datasets. Our approach partitions the data with the help of an adaptive grid and applies a sequence of steps implemented by optimized processing modules that minimize the required comparisons. We showed how our grid-based approach scales out by dispatching stripes of cells to load-balanced Spark jobs, exploiting parallelism. Finally, we proposed an approximate algorithm with accuracy guarantees, which is one order of magnitude faster compared to our exact method.

Future work includes adapting the method to other rank-based correlation measures, as well as developing grid-based exact and approximate solutions to other multidimensional analysis problems that count rank statistics, such as top-k dominating queries [40].

ACKNOWLEDGEMENTS

Work supported by project MIS 5154714 of the National Recovery and Resilience Plan Greece 2.0 funded by the European Union under the NextGenerationEU Program.

REFERENCES

[1] Asaf Adi, David Botzer, Opher Etzion, and Tali Yatzkar-Haham. 2001. Monitoring Business Processes through Event Correlation based on Dependency Model. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, CA, USA, May 21-24, 2001*. ACM, 610.

[2] Foto N. Afrati, Paraschos Koutris, Dan Suciu, and Jeffrey D. Ullman. 2012. Parallel skyline queries. In *15th International Conference on Database Theory, ICDT '12, Berlin, Germany, March 26-29, 2012*. ACM, 274–284.

[3] David G. Aragones, Miguel Palomino-Segura, Jon Sicilia, Georgiana Crainiciuc, Iván Ballesteros, Fátima Sánchez-Cabo, Andrés Hidalgo, and Gabriel F. Calvo. 2024. Variable selection for nonlinear dimensionality reduction of biological datasets through bootstrapping of correlation networks. *Computers in Biology and Medicine* 168 (2024), 107827.

[4] Gaia ESA Archive. 2022. Gaia DR3 dataset. https://cdn.gea.esac.esa.int/Gaia/gdr3/gaia_source/.

[5] Stephan Arndt, Carolyn Turvey, and Nancy C. Andreasen. 1999. Correlating and predicting psychiatric symptom ratings: Spearmans r versus Kendalls tau correlation. *Journal of Psychiatric Research* 33, 2 (1999), 97–104.

[6] Daniel F. Brossart, Vanessa C. Laird, and Trey W. Armstrong. 2018. Interpreting Kendall's Tau and Tau-U for single-case experimental designs. *Cogent Psychology* 5, 1 (2018), 1518687.

[7] Dar-Jen Chang, Ahmed H. Desoky, Ming Ouyang, and Eric C. Rouchka. 2009. Compute Pairwise Manhattan Distance and Pearson Correlation Coefficient of Data Points with GPU. In *10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing, SNPD 2009*. IEEE Computer Society, 501–506.

[8] Shiyang Chen, Amin Ghadami, and Bogdan I. Epureanu. 2022. Practical Guide of Using Kendall's τ in the Context of Forecasting Critical Transitions. *Royal Society Open Science* 9 (2022). Issue 7.

[9] Dave Dhruvil. 2021. Internet Speed Dataset. https://www.kaggle.com/datasets/dhruvildave/ookla-internet-speed-dataset/.

[10] Christos Doulkeridis, Akrivi Vlachou, Dimitris Mpestas, and Nikos Mamoulis. 2017. Parallel and Distributed Processing of Spatial Preference Queries using Keywords. In *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017*. OpenProceedings.org, 318–329.

[11] Taban Eslami, Muaaz Gul Awan, and Fahad Saeed. 2017. GPU-PCC: A GPU Based Technique to Compute Pairwise Pearson's Correlation Coefficients for Big fMRI Data. In *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics, BCB 2017, Boston, MA, USA, August 20-23, 2017*. ACM, 723–728.

[12] National Centers for Environmental Information. 2025. Climatological Data Archive. https://www.ncei.noaa.gov/data/local-climatological-data/archive/.

[13] M. R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

[14] Nguyen Ho, Torben Bach Pedersen, Van Long Ho, and Mai Vu. 2020. Efficient Search for Multi-Scale Time Delay Correlations in Big Time Series Data. In *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020*. OpenProceedings.org, 37–48.

[15] Chun-Ting Yang Jie-Huei Wang. 2022. Identification of Gene-Environment Interactions by Non-Parametric Kendall's Partial Correlation with Application to TCGA Ultrahigh-Dimensional Survival Genomic Data. *Frontiers in Bioscience-Landmark* 27, 8 (2022), 225.

[16] Puloma Katiyar, Tin Vu, Ahmed Eldawy, Sara Migliorini, and Alberto Belussi. 2020. SpiderWeb: A Spatial Data Generator on the Web. In *SIGSPATIAL '20: 28th International Conference on Advances in Geographic Information Systems, Seattle, WA, USA, November 3-6, 2020*. ACM, 465–468.

[17] Maurice G Kendall. 1938. A new measure of rank correlation. *Biometrika* 30, 1-2 (1938), 81–93.

[18] Hideaki Kimura, George Huo, Alexander Rasin, Samuel Madden, and Stanley B. Zdonik. 2009. Correlation Maps: A Compressed Access Method for Exploiting Soft Functional Dependencies. *Proc. VLDB Endow.* 2, 1 (2009), 1222–1233.

[19] William R. Knight. 1966. A Computer Method for Calculating Kendall's Tau with Ungrouped Data. *J. Amer. Statist. Assoc.* 61, 314 (1966), 436–439.

[20] Nikolaos Koutroumanis, Christos Doulkeridis, and Akrivi Vlachou. 2025. Parallel Spatial Join Processing with Adaptive Replication. In *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025*. OpenProceedings.org, 464–476.

[21] Mirella Lapata. 2006. Automatic Evaluation of Information Ordering: Kendall's Tau. *Comput. Linguistics* 32, 4 (2006), 471–484.

[22] Pierre Legendre. 2005. Species Associations: The Kendall Coefficient of Concordance Revisited. *Journal of Agricultural, Biological, and Environmental Statistics* 10, 2 (2005), 226–245.

[23] Hanwen Liu, Mihail Stoian, Alexander van Renen, and Andreas Kipf. 2024. Corra: Correlation-Aware Column Compression. In *Proceedings of Workshops at the 50th International Conference on Very Large Data Bases, VLDB 2024, Guangzhou, China, August 26-30, 2024*. VLDB.org.

[24] Stavros Maroulis, Nikos Bikakis, George Papastefanatos, Panos Vassiliadis, and Yannis Vassiliou. 2021. RawVis: A System for Efficient In-situ Visual Analytics. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. ACM, 2760–2764.

[25] Koen Minartz, Jens E. d'Hondt, and Odysseas Papapetrou. 2022. Multivariate correlations discovery in static and streaming data. *Proc. VLDB Endow.* 15, 6 (2022), 1266–1278.

[26] Yashuang Mu, Xiaodong Liu, and Lidong Wang. 2018. A Pearson's correlation coefficient based decision tree and its parallel implementation. *Inf. Sci.* 435 (2018), 40–58.

[27] Kasper Mullesgaard, Jens Laurits Pederseny, Hua Lu, and Yongluan Zhou. 2014. Efficient Skyline Computation in MapReduce. In *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24-28, 2014*. OpenProceedings.org, 37–48.

[28] Divya Pandove and Avleen Malhi. 2021. A Correlation Based Recommendation System for Large Data Sets. *J. Grid Comput.* 19, 4 (2021), 42.

[29] Yoonjae Park, Jun-Ki Min, and Kyuseok Shim. 2013. Parallel Computation of Skyline and Reverse Skyline Queries Using MapReduce. *Proc. VLDB Endow.* 6, 14 (2013), 2002–2013.

[30] Tilemachos Pechlivanoglou, Mahmoud Alsaeed, and Manos Papagelis. 2020. MRSweep: Distributed In-Memory Sweep-line for Scalable Object Intersection Problems. In *7th IEEE International Conference on Data Science and Advanced Analytics, DSAA 2020, Sydney, Australia, October 6-9, 2020*. IEEE, 324–333.

[31] Llukan Puka. 2011. Kendall's Tau. In *International Encyclopedia of Statistical Science*. 713–715.

[32] Yasushi Sakurai, Spiros Papadimitriou, and Christos Faloutsos. 2005. BRAID: Stream Mining through Group Lag Correlations. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*. ACM, 599–610.

[33] Petr Socha, Vojtech Miskovský, Hana Kubátová, and Martin Novotný. 2017. Optimization of Pearson correlation coefficient calculation for DPA and comparison of different approaches. In *20th IEEE International Symposium on Design and Diagnostics of Electronic Circuits & Systems, DDECS 2017, Dresden, Germany, April 19-21, 2017*. IEEE, 184–189.

[34] Dimitrios Tsitsigkos, Panagiotis Bouros, Nikos Mamoulis, and Manolis Terrovitis. 2019. Parallel In-Memory Evaluation of Spatial Joins. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL*. ACM, 516–519.

[35] Akrivi Vlachou, Christos Doulkeridis, and Yannis Kotidis. 2008. Angle-based space partitioning for efficient parallel skyline computation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD*. ACM, 227–238.

[36] Chayse Wright. 2021. National Solar Radiation Data. https://www.kaggle.com/datasets/chaysewright/national-solar-radiation-database-19912005.

[37] Yingjun Wu, Jia Yu, Yuanyuan Tian, Richard Sidle, and Ronald Barber. 2019. Designing Succinct Secondary Indexing Mechanism by Exploiting Column Correlations. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. ACM, 1223–1240.

[38] Hui Xiong, Shashi Shekhar, Pang-Ning Tan, and Vipin Kumar. 2004. Exploiting a support-based upper bound of Pearson's correlation coefficient for efficiently identifying strongly correlated pairs. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*. ACM, 334–343.

[39] Weichao Xu, Yunhe Hou, Y. S. Hung, and Yuexian Zou. 2013. A comparative analysis of Spearman's rho and Kendall's tau in normal and contaminated normal models. *Signal Processing* 93, 1 (2013), 261–276.

[40] Man Lung Yiu and Nikos Mamoulis. 2009. Multi-dimensional top-k dominating queries. *VLDB J.* 18, 3 (2009), 695–718.

[41] Aoqian Zhang, Zexue Wu, Yifeng Gong, Ye Yuan, and Guoren Wang. 2024. Multivariate Time Series Cleaning under Speed Constraints. *Proc. ACM Manag. Data* 2, 6 (2024), 245:1–245:26.

[42] Maoqing Zhang, Wuzhao Li, Liang Zhang, Hao Jin, Yashuang Mu, and Lei Wang. 2023. A Pearson correlation-based adaptive variable grouping method for large-scale multi-objective optimization. *Inf. Sci.* 639 (2023), 118737.

[43] Yu Zhang, Youzhong Ma, and Xiaofeng Meng. 2014. Efficient Spatio-textual Similarity Join Using MapReduce. In *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), Warsaw, Poland, August 11-14, 2014 - Volume II*. IEEE Computer Society, 52–59.

[44] Zichen Zhu, Xiao Hu, and Manos Athanassoulis. 2023. NOCAP: Near-Optimal Correlation-Aware Partitioning Joins. *Proc. ACM Manag. Data* 1, 4 (2023), 252:1–252:27.