RELATE: Preference-aware Correlation-based Query Refinement

Abdullah Albarrak
Al Imam Mohammad Ibn Saud Islamic University
Saudi Arabia
amsbarrak@imamu.edu.sa

ABSTRACT: *Data exploration techniques aim to efficiently and effectively guide users towards interesting data within massive, complex data. Being one of those techniques, query refinement goal is to automatically refine a user's query so that its result satisfies a certain constraint. In this paper, we focus on time series correlation as a constraint for refinement such that the result of a query is a pair of two subsequence time series. Differently than most existing works, we make use of the deviation from a correlation constraint as an objective to minimize in our problem. Moreover, we include users preferences as an objective to maximize in proportion to users input queries. The combination of these two objectives are prevalent in applications where users seek time series subsequences that exhibit a certain correlation value range and are close from their initial queries. Towards finding the optimal query satisfying these objectives, we propose an efficient computational-centric algorithm and an innovative I/O-centric algorithm as well. We experimentally validate the efficiency and effectiveness of our proposed algorithms using real and synthetic data compared to recently proposed correlation-based subsequence search algorithms.*

Keywords: Time Series, Correlation, Query Refinement

Received: 10 April 2019, Revised 25 August 2019, Accepted 5 September 2019

© 2019 DLINE. All Rights Reserved

DOI: 10.6025/jcl/2019/10/4/108-125

1. Introduction

Data nowadays is in continuous increase in terms of size and complexity. This increase easily overwhelms data users when they start exploring these ever-growing datasets. Hence, the need for data exploration techniques which guide users towards interesting areas within massive, complex data has increased in parallel with the increase of data's volume and complexity [4, 6, 9, 7, 2].

Query refinement is one of those techniques where users queries are automatically refined so that queries results satisfy certain constraints [13, 18], e.g., aggregate constraints [19, 11, 1]. Without the support of query refinement techniques, users can do nothing but to manually refine the query, submit it, inspect the result, then refine again if the result does not satisfy a certain constraint. This labor intensive task leads to users frustration, and adds unnecessary overload to the database system [20].[19, 11, 1]. Without the support of query refinement techniques, users can do nothing but to manually refine the query, submit it, inspect the result, then refine again if the result does not satisfy a certain constraint. This labor intensive task leads to users frustration, and adds unnecessary overload to the database system [20].

Time series data are one type of complex data. It is being generated, gathered and stored in unprecedented rate, whether for the purpose of financial analysis (e.g., exchange rates, stock market), environment monitoring (e.g., humidity, wind), or network traffic analysis (e.g., servers loads, incoming TCP requests). One key task of time series exploration is the search for correlated subsequences [10, 14, 17, 21], where correlation is the Pearson Correlation Coefficient.

We model this exploratory task as a query refinement problem with Pearson correlation as a user specified constraint. Therefore, the goal is to refine a query so that its result (which are two subsequences) satisfies a given correlation value. To find that refined query, a solution has to iterate over all possible combinations of synchronized sub- sequences. Interestingly, this exploratory task is not trivial, it is actually CPU and I/O intensive. For example, given two time series with equal length n, there are $n(n+1)/2$ combinations of subsequences to be examined. If the cost for computing a subsequence correlation is C, then the time complexity of this task is $O(Cn^2)$. Hence, several works have focused on optimizing this task using complex indexing methods and pruning techniques.

For instance, SKIP [10] was proposed to find the longest subsequence of two time series having a correlation above a threshold value θ, without any prior knowledge of the subsequence length m. Jocor [14] focus on finding the subsequence with the highest correlation value such that it has a length above a threshold value ml. While the above works are successful in providing users with results satisfying their thresholds, they fall short in addressing correlation thresholds as ranges (i.e., correlation between θ_1 and θ_2) and users preferences. The following example illustrates more:

Example 1. (Stocks Market): Investment portfolios in stocks market aim to maximize prot by managing (reducing) risks in the invested assets. One way to reduce risks is to split investments between non-correlated assets so that when one asset takes a loss, others most likely will not follow.

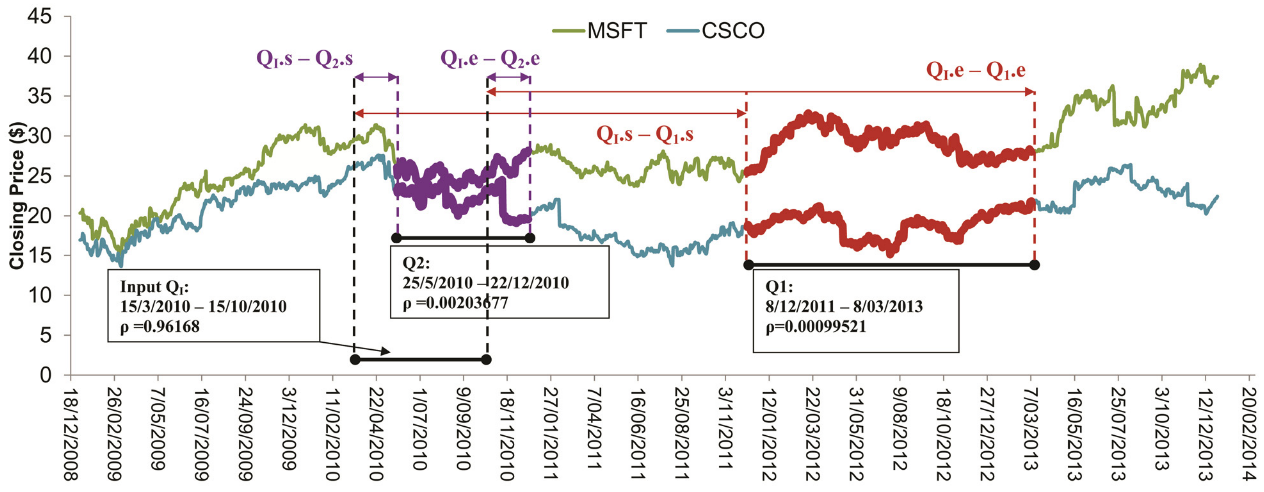


Figure 1. Closing prices for two stocks MSFT and CSCO extracted from Yahoo! Finance for the period 18/12/2008 to 20/2/2014

Given a database of stock prices history[1] of US companies in the stock market for the period of 2-1-2009 to 31-12-2013, an investor wants to know what was the best time interval to buy stocks from Microsoft Corporation (MSFT) and Cisco Systems (CSCO) such that the risk of loss is negligible (i.e., correlation is within a very low range [0 –0.1]). By extracting this information,

[1]e.g., data from http://nance.yahoo.com/

she can build a future plan to buy stocks from MSFT and CSSO in a specific time interval hoping that her portfolios will grow with minimum risks (with the assumption of a stable financial stocks market). Further, let us assume that she is limited to buy stocks within a certain time interval, hence, she has a preference over time as well. She starts exploring this database and query for the subsequences between 15/03/2010 and 15/10/2010 as shown in Figure 1. She is surprised that the correlation of Q_I is $\rho =$ 0.96168, i.e., it is too far from the range [0 – 0.1]. So, she changes the time interval (per her preference) and resubmit the query. She keeps going through those steps until finding a query that returns two subsequences with a correlation value as close as possible to her range, within the time interval.

In Example 1, the investor wants to buy stocks from MSFT and CSCO within her initial time interval (we call this interval: user's preference) such that there is a low risk of loss, i.e., correlation is within [0 –0.1]. So, she asks: what was the best time interval in which if I bought stocks from MSFT and CSCO around 15/03/2010 - 15/10/2010, my risk of loss is minimal? There are two ways to answer this question. First, to ignore her preference (i.e., the period 15/10/2010 - 15/03/2010) and apply the SKIP algorithm to search for a low correlation subsequences. The SKIP algorithm would return the red-colored subsequences (query Q_1), as illustrated in Figure 1. Query Q_1 have a correlation of $\rho = 0.00099521$. Although Q_1 exhibits a low correlation value, it is too far from her preference, which would leave the answer to be of low-benefit. Second, to include her preference when searching for the subsequences, which would result in the purple- colored subsequences (query Q_2) to be the answer.

Employing users' preferences in correlation based subsequences time series search problem promotes optimization opportunities with guarantees on the result's accuracy. Plainly, our proposed optimization techniques strive to reduce CPU and I/O cost through harnessing users' prefer users' preferences to limit the search space, and prune unpromising candidate queries by estimating their correlation values in an efficient manner. We summarize our contributions as follows:

Incorporating users preferences and target correlation ranges as objectives for preference-aware correlation based query refinement (RELATE) problem.

• Proposing an efficient CPU-centric algorithm to find optimal queries by utilizing: a) users preferences to limit, navigate and prune the search space, and b) multiple copies of cumulative arrays which summarize time series values.

• Implementing an innovative and efficient I/O-centric algorithm which estimates queries' correlation values by means of a simple yet efficient meta-data collected offline for each time series.

• Conducting experiments on real and synthetic data to evaluate our algorithms and show the efficiency they provide when compared to recent algorithms for subsequence time series search.

2. Preliminaries and Definitions

We assume there are k data series $x_1, x_2, ... x_k$ stored in a database such that all series are of equal length. A data series x_i is a series of n consecutive values $x_i = \{v_1, v_2, ... v_n\}$ that have an implicit ordering. For instance, we assume x_i is a time series such that its values ordering is based on a timestamp domain (e.g., date and/or time). While our work is based on time series, it can be generalized to data series too.

A subsequence of x_i is constructed by a time interval $[s, e]$, $x_i[s, e] = \{v_j, v_{j+1}, ... v_e\}$ where $j = s$, $s < e$ and $e \leq n$, as shown in Figure 2.

Our focus in this paper is on correlation as a constraint. To compute the correlation, there should be two time series. Hence, users explore time series based on a pair of synchronized subsequences $x_i[s, e]$ and $x_j[s, e]$ constructed from time series x_i and x_j, respectively. Henceforth, we refer to the pair $x_i[s, e]$ and $x_j[s, e]$ as a candidate query $Q_c(x_i, x_j, s, e)$ or briefly Q_c when there is no need to specify which time series and what time interval.

As discussed in Example 1, users explore time series to find subsequences within a certain correlation range. In other words, they prefer subsequences that have the minimum deviation from their target correlation range. We explain next how to support this objective in details.

Symbol	Definition
x_i	A time series
n	Length of time series
m	Length of subsequence time series
$x_i[s, e]$	A subsequence of x_i
λ	Weight of preference
tc	Target correlation range $[c_l - c_u]$
Q_I	Input query
Q_c	Candidate query
$\Delta^{\rho}_{Q_c}$	Correlation deviation of Q_c
$\Delta^{E}_{Q_c}$	Preference deviation of Q_c
ΔQ_c	Total deviation of Q_c

Table 1. Symbols and Definitions

2.1. Correlation Deviation

The deviation in Δ^{ρ} correlation for a candidate query Q_c from a target correlation range $tc = [c_1 - c_u]$ is a measure of how far the correlation value of the subsequences returned by Q_c is to the target range. Formally:

$$\Delta^{\rho}_{Q_c} = \begin{cases} 0 & \text{if } \rho(Q_c) \in [c_l - c_u] \\ \rho(Q_c) - c_l & \text{if } \rho(Q_c) < c_l \\ \rho(Q_c) - c_u & \text{if } \rho(Q_c) > c_u \end{cases} \quad (1)$$

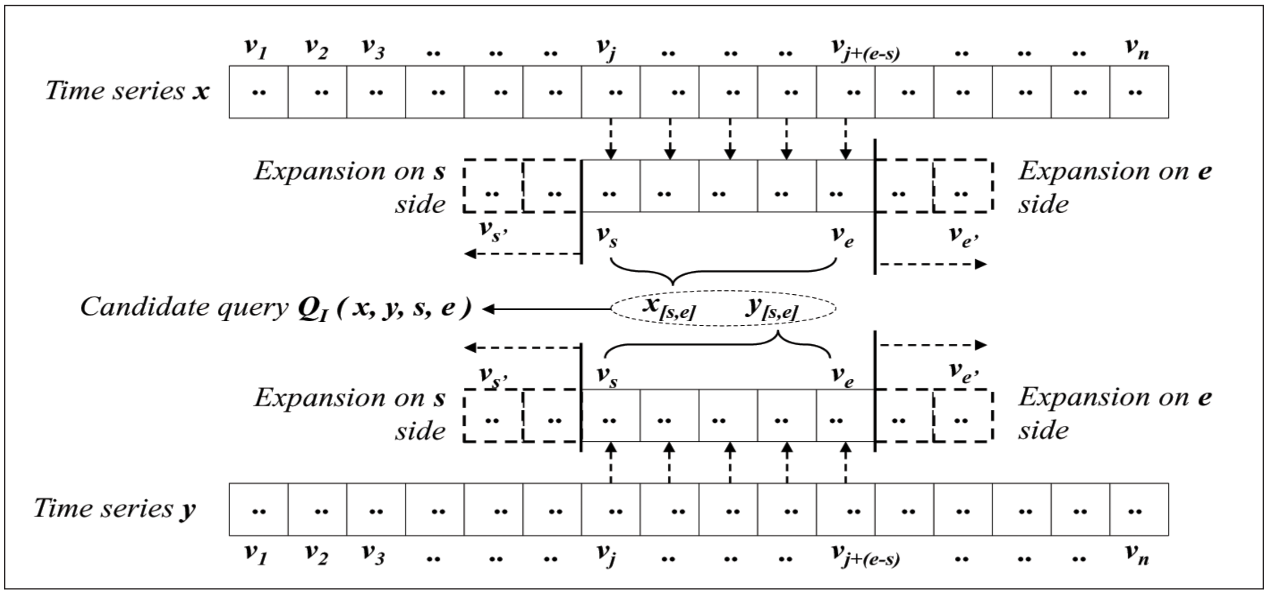
where $\rho\,(Q_c)$ is a function that returns the Pearson correlation coefficient of the subsequences returned by Q_c. When $\Delta^{\rho}{}_{Qc}$ equals zero it means Q_c's correlation value is within the target correlation range tc. Otherwise, it is the deviation from the closest sides of tc to $\rho\,(Q_c)$.

The Pearson correlation coefficient of two subsequences x and y (returned by Q_c) of length m is computed as follows [10]:

$$\rho(Q_c) = \rho(x, y) = \frac{\sum_{i=1}^{m} x_i y_i - \sum_{i=1}^{m} x_i \sum_{i=1}^{m} y_i}{\sqrt{\sum_{i=1}^{m} x_i^2 - (\sum_{i=1}^{m} x_i)^2}\sqrt{\sum_{i=1}^{m} y_i^2 - (\sum_{i=1}^{m} y_i)^2}} \quad (1)$$

Example 2. Query Q_I, Q_1 and Q_2 in Figure 1 have a correlation values of 0.96168, 0.00099521 and 0.00203677 respectively. Consequently, Q_1 and Q_2 equal to zero when $tc = [0 - 0.1]$ while $\Delta^{\rho}_{Q_I}$ equals 0.86168.

In Example 2, both Q_1 and Q_2 have zero deviation, but one of them is more similar and closer to the user's initial query Q_I. Specifically, from Figure 1, it is clear that Q_2 is indeed closer to Q_I than Q_1. Hence, suggesting Q_2 as a refined query to the user is more beneficial. But how to obtain Q_2 (or any candidate query Q_c) in the first place? We explain the how next.



Time series x
Time series y
v_1 v_2 v_3 v_j $v_{j+(e-s)}$ v_n
Expansion on s side
Expansion on e side
$v_{s'}$ v_s v_e $v_{e'}$
Candidate query Q_I (x, y, s, e)
$x_{[s,e]}$ $y_{[s,e]}$

Figure 2. Refinement of $Q_I(x,\ y, s, e)$ by expanding either sides s or e, i.e., $\mathcal{LE}$, $\mathcal{RE}$

2.2 Query Refinement

Refining $Q_I(x, y, s, e)$ will produce a refined candidate query $Q_c(x, y, \hat{s}, \hat{e})$ which has a new time interval $[\hat{s}, \hat{e}]$ that might have an overlap with Q_I's interval $[s, e]$. Specifically, there are two refinement operations applied on the time interval $[s, e]$, as shown in Figure 2.

1. Expansion: To expand $[s, e]$ from either sides s or e by δ. For instance, $[\hat{s}, e]$ is expanded from s side by δ such that $\hat{s} = s - \delta$ while $[s, \hat{e}]$ is expanded from e side by δ such that $\hat{e} = e + \delta$. We encode those two operations as $\mathcal{LE}$ (left expansion) and $\mathcal{RE}$ (right expansion).

2. Contraction: To contract $[s, e]$ from either sides s or e by δ. For instance, $[\hat{s}, e]$ is contracted from s side by δ such that $\hat{s} = s + \delta$ while $[s, \hat{e}]$ is contracted from e side by δ such that $\hat{e} = e - \delta$. Similarly, we encode those two operations as $\mathcal{LC}$(left contraction) and $\mathcal{RC}$ (right contraction).

With those two refinement operations, we are able to generate all possible $\frac{n(n+1)}{2}$ combinations of subsequences (or equivalently: candidate queries). To remove any approximation and to ensure no possible candidate query is missed, we set $\delta = 1$.

Now, the question is, how to judge whether a candidate query Q_c is more beneficial than another one? We answer this question by quantifying the benefit of a query as a preference deviation, which is explained next.

2.3. Preference Deviation

Since users prefer similar queries to their first impression (i.e., Q_I), we include users preferences in the objective of refinement as a preference deviation. It is a normalized value that indicates how far Q_c is from the input query Q_I. A query that is far from a user's preference will exhibit a high deviation, and vise versa. Because a query Q_c is defined using a time interval, i.e., $[s, e]$, only this time interval is involved in defining the preference deviation $\Delta^S_{Q_c}$:

$$\Delta^E_{Q_c} = \frac{|Q_I.s - Q_c.s|}{n} + \frac{|Q_I.e - Q_c.e|}{n} \quad (3)$$

where n is the length of the two time series. Note that the lower the value of $\Delta^E_{Q_c}$, the more beneficial Q_c is to the user.

Example 3. Query Q_1 and Q_2 in Figure 1 have a preference deviation of 0.826 and 0.0763 respectively, hence, Q_2 is more beneficial to Q_I then Q_1 .

Algorithm 1 Baseline (simplified SKIP)

```
1: **Input:** Input query $Q_I(x, y, s, e)$, preference weight $\lambda$,
   target correlation $tc$, minimum length $ml$, $\alpha$
2: **Return:** $Q_p$, $best$;
3: calculate $S^\alpha_x$, $S^\alpha_{x^2}$, $S^\alpha_y$, $S^\alpha_{y^2}$ and $S^\alpha_{xy}$ for whole time series
   $x, y$
4: $best = 1$; $Q_p \leftarrow Q_I$
5: for ( $l = n$ to $ml$ ) do
6:    for ( $t = 0$ to $n - l$ ) do
7:       $Q_i \leftarrow (x, y, t, (t + l - 1))$;
8:       for ( $z \in \{x, y, x^2, y^2, xy\}$) do
9:          if ( $t = 0$ ) then
10:            Compute $\sum z$ by $S^\alpha_z$
11:         else
12:            Update $\sum z$ incrementally
13:      Compute $\rho(Q_i)$ ;
14:      $\Delta_{Q_i} = \lambda\Delta^E_{Q_i} + (1 - \lambda)\Delta^\rho_{Q_i}$;
15:      if ( $\Delta_{Q_i} < best$ ) then
16:         $best = \Delta_{Q_i}$ ; $Q_p \leftarrow Q_i$;
17: return $Q_p$, $best$;
```

2.4. Problem Definition

Now, we are in position to formally define the problem of preference-aware correlation-based query refinement (RE- LATE):

Definition 1. Given $Q_I(x, y, s, e)$, an input query to retrieve two subsequences $x[s, e]$ and $y[s, e]$, and a target correlation range tc. Find the optimal query $Q_p(x, y, \hat{s}, \hat{e})$ such that Q_p minimizes the deviation in correlation $\Delta^\rho_{Q_p}$ while considering users preferences by minimizing the preference deviation $\Delta^E_{Q_p}$.

As stated in Definition 1, the objective of RELATE is to look for the query Q_p that minimizes the overall deviation defined as:

$$\Delta_{Q_p} = \lambda\Delta^S_{Q_p} + (1 - \lambda)\Delta^\rho_{Q_p} \tag{4}$$

The parameter λ is used to control the trade o between satisfying the correlation and preference deviation. Setting $\lambda = 0$ means users preferences are neglected, which is a special case of our general objective.

Example 4. Continuing the stock market example, Query Q_1 and Q_2 have an overall deviation of 0.413 and 0.03815 respectively, when $\lambda = 0.5$.

Table 1 summarizes the rest of the parameters and symbols used throughout this paper.

3. Methodology

The refinement problem RELATE is essentially a search problem. That is, to find Q_p, an algorithm has to iterative over all possible combinations of subsequences (candidate queries), compute the objective Eq. (4), while keeping the one with the minimum deviation. Iterating over all candidate queries is needed because approximation is not an option.

Iterating over all candidate queries obviously incurs high overhead with regard to both angles: CPU and I/O. Hence, we tackle the RELATE problem from these angles individually, and propose computational-centric (Section 3.2) and I/O-centric (Section 3.3) algorithms with efficient optimization techniques.

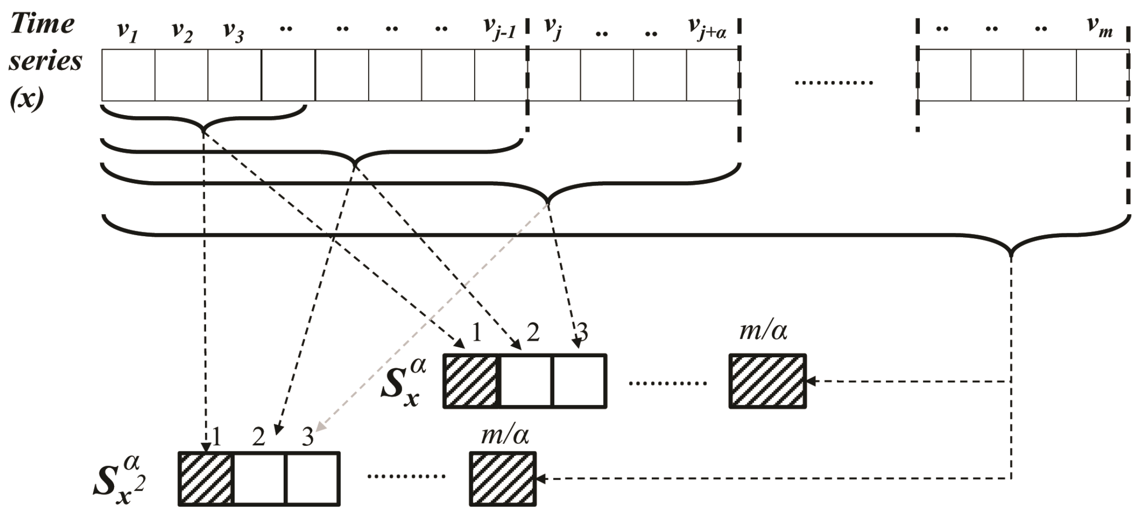


Figure 3. Constructing -cumulative sum arrays $S^a_{\ x}$ and $S^\alpha_{x^2}$ for time series x of length $m.\alpha = 4$

3.1 Computations of Correlation

Computations cost of correlation in Eq. 2 increases linearly with n (i.e., complexity = $\mathcal{O}(n)$). Specifically, each of the summation components in Eq. 2 will perform n summation operations. With the observation that Eq. 2 can be computed incrementally [17], [10] proposed the -skipping cumulative array to compute it in $\mathcal{O}(\alpha)$ time, where $\alpha \lll n$.

In the -skipping cumulative array [10], each time series x of length n has two cumulative arrays; the sum of values and the sum of square values, S^α_x and $S^\alpha_{x^2}$, resp. Those arrays are of length $\frac{n}{\alpha}$. An element in those arrays is computed as follows (see Figure 3):

$$S^\alpha_x[j] = \sum_{i=1}^{j*\alpha} x[i], \text{ if } (j*\alpha) \bmod \alpha = 0$$

$$S^\alpha_{x^2}[j] = \sum_{i=1}^{j*\alpha} (x[i])^2, \text{ if } (j*\alpha) \bmod \alpha = 0$$

$$\text{where } j = 1, 2, ..., \frac{n}{\alpha}$$

The cumulative sum of product values array S^α_{xy} of two time series x and y is generated in a similar way:

$$S^\alpha_{xy}[j] = \sum_{i=1}^{j*\alpha} x[i]*y[i], \text{ if } (j*\alpha) \bmod \alpha = 0$$

$$\text{where } j = 1, 2, ..., \frac{n}{\alpha}$$

Hence, with those -skipping cumulative arrays, the components in Eq. 2 are computed in $\mathcal{O}(\alpha)$ time.

Next, we describe two algorithms which utilize the α- skipping cumulative arrays to find Q_p. Then, we show our algorithm which incorporates the preference objective to optimize the search for Q_p without sacrificing the solution quality.

3.2. Computational-centric Algorithms

In this section we discuss three algorithms to find the optimal solution Q_p. We start by showing a brute force algorithm that we consider as a baseline to compare against the other algorithms.

Baseline: This is a simplified version of SKIP [10] which was proposed to find the longest correlated subsequence of

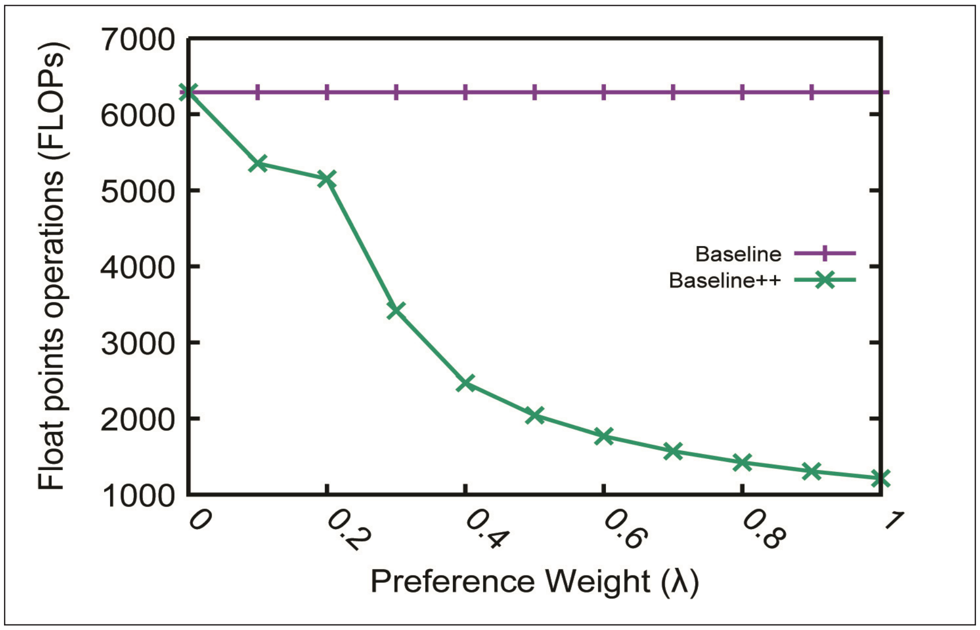


Figure 4. Cost construction of cumulative arrays of both algorithms for a time series x of length $n = 1258$, $\alpha = 8$

multiple time series. We simplified and adapted the algorithm for two time series refinement, i.e., $O = \{x, y\}$.

Essentially, Alg. 1 utilizes two nested for-loops (lines 5,6) to generate the combinations of subsequences. The outer- loop species the right hand side of the time interval, while the inner-loop specifics the left hand side. In each iteration, the interval $[t, (t + 1 - 1)]$ corresponds to the time interval $[s, e]$, which is used to represent a possible query Q_i (line 7).

Steps in lines 8-12 are the core of SKIP [10] for computing correlation: If the left hand side equals zero (line 9), then correlation is computed by the cumulative arrays S_x^α, $S_{x^2}^\alpha$, S_y^α, $S_{y^2}^\alpha$ and S_{xy}^α that are generated for the whole time series in advance (line 3). This result in a complexity of $O(\alpha)$ only.

Then, the values computed from the previous step are updated incrementally (line 12) to compute correlation for the next candidate queries, until the inner-loop finishes. Specifically, the next candidate query is generated by shifting the interval to the right by one step, which means adding one value to the right hand side, and subtracting one value from the left. Lastly, lines 14-16 are added to support our objective in refinement.

Baseline++ (limited search space): Differently than the previous algorithm, Baseline++ limits the search space before navigating. Hence, it searches a small part of the time series instead of the whole time series. To achieve that, it uses the preference objective to limit the search space (lines 5-10).

Specifically, it defines two variables maxs and maxe as new boundaries for the search space. Then, starting from QI ‘s boundaries (line 4), it enters a while-loop (line 5) until: 1) both sides of the time series have been reached, or 2) the preference deviation reached its maximum possible value (lines 7, 8).

There are two prominent differences between Baseline and Baseline++. Firstly, the generation of the cumulative arrays. The cumulative arrays in Baseline++ are generated for a sub time interval of x and y, which are essentially the new found boundaries [*maxs, maxe*] (line 10), not the whole interval as in Baseline. Figure 4 shows how much Baseline++ can save in terms of construction cost of the cumulative arrays when compared to Baseline. For example, when $\lambda = 0.5$, Baseline++ saves %68 of construction cost. However, the maximum ratio for the construction cost of the cumulative arrays to the cost of refinement for Baseline and Baseline++ is approximately $\approx$1:7000, hence, there is more need to improve in the refinement side.

Algorithm 2 Baseline++ (limited search space)

```
1: Input: Input query $Q_I(x, y, s, e)$, preference weight $\lambda$,
   target correlation $tc$, minimum length $ml$, $\alpha$
2: Return: $Q_p$, $best$;
3: $best$ = (1-$\lambda$)$\Delta^{\rho}_{Q_I}$; $Q_p \leftarrow Q_I$;
4: $i = Q_I.s$; $j = Q_I.e$;
5: while ( $i \geq 0 \,||\, j \leq n$ ) do
6:    $Q_i \leftarrow$(i,j);
7:    if ( $\lambda\Delta^{E}_{Q_i} > best$ ) then
8:       $maxs = i; maxe = j$; break;
9:    $i \leftarrow i - 1$; $j \leftarrow j + 1$;
10: calculate $S^{\alpha}_{x}$, $S^{\alpha}_{x^2}$, $S^{\alpha}_{y}$, $S^{\alpha}_{y^2}$ and $S^{\alpha}_{xy}$ for time interval
    $[maxs, maxe]$ of $x, y$
11: $n = maxe - maxs + 1$;
12: for ( $l = n$ to $ml$ ) do
13:    for ( $t = maxs$ to $n - l$ ) do
14:       $Q_i \leftarrow (x, y, t, (t + l - 1))$;
15:       for ( $z \in \{x, y, x^2, y^2, xy\}$) do
16:          if ( $t = maxs$ ) then
17:             Compute $\sum z$ by $S^{\alpha}_{z}$
18:          else
19:             Update $\sum z$ incrementally
20:       Compute $\rho(Q_i)$ ;
21:       $\Delta_{Q_i} = \lambda\Delta^{E}_{Q_i} + (1 - \lambda)\Delta^{\rho}_{Q_i}$;
22:       if ( $\Delta_{Q_i} < best$ ) then
23:          $best = \Delta_{Q_i}$ ; $Q_p \leftarrow Q_i$;
24: return $Q_p$, $best$;
```

Secondly, the new boundaries [maxs, maxe] in Baseline++ will decrease the number of candidate queries (when $\lambda > 0$). As a result, the cost of refinement will decrease too. Though, Baseline++'s technique in optimizing the refinement cost is limited by the query/time series length ratio, as we shall see in the experiments.

Next, we introduce our algorithm Incremental (INC) which uses the preference deviation as an optimization technique to prune unpromising candidate queries. INC further optimize the computations of correlation by a simple technique: creating and maintaining four instances of the cumulative arrays, one for each refinement operation: $\mathcal{LE}, \mathcal{RE}, \mathcal{LC}, \mathcal{RC}$. Incremental (*INC*): Essentially, INC starts from Q_I to generate all possible subsequences with the help of an auxiliary function Refine() and a priority queue $\mathcal{Q}$. This auxiliary function takes a query as an input, and performs the four refinement operations defined earlier in Section 2.2 to produce the set Q_c. Then, for each query $Q_i \in Q_c$, the corresponding cumulative arrays instance are updated incrementally. After that, the correlation is computed from this instance, and Q_i is pushed into $\mathcal{Q}$. Once all candidate queries in Q_c have been processed, INC pops an unrefined query Q_i from $\mathcal{Q}$ and calls Refine(), and so on. At any time, the candidate queries in $\mathcal{Q}$ are ascending sorted from the closest to Q_I to the furthest based on Eq. 3.

To avoid an exhaustive search, INC abandons the search when reaching a point where any candidate query to be generated has a preference deviation higher than the best deviation found so far, or when the queue becomes empty, as shown in Alg. 3 line 7.

However, INC assumes that both time series being explored are loaded in memory in advance. This assumption though is not practical for long time series from the I/O cost point of view. Hence, we discuss next our I/O-centric algorithm which iteratively retrieves a small parts of the time series (i.e., a candidate query Q_c) via a DBMS call if and only if Q_c has a high chance to be the optimal solution. Otherwise, if Q_c has little chance to be an optimal solution, the algorithm will prune it, resulting in saving I/O cost.

Algorithm 3 Incremental (INC)

```
1: Input: Input query $Q_I(x, y, s, e)$, preference weight $\lambda$
2: Return: $Q_p$, $best$;
3: Define and compute $S_x^{\mathcal{A}}$, $S_{x^2}^{\mathcal{A}}$, $S_y^{\mathcal{A}}$, $S_{y^2}^{\mathcal{A}}$, $S_{xy}^{\mathcal{A}}$ for $[s, e]$
4: where $\mathcal{A} \in \{\mathcal{RE}, \mathcal{RC}, \mathcal{LE}, \mathcal{LC}\}$
5: $best$ = (1-$\lambda$)$\Delta_{Q_I}^{\rho}$; $Q_p \leftarrow Q_I$;
6: $\mathcal{Q}$.push($Q_I$);
7: while ( $\mathcal{Q} \neq \phi$ & $\Delta_{TH} \leq best$) do
8:    $Q_c \leftarrow$ Refine($\mathcal{Q}$.pop());
9:    for ( each $Q_i \in Q_c$ s.t. $Q_i$ not visited ) do
10:       Lookup $\mathcal{A}$ of $Q_i$ , i.e., $\{\mathcal{RE}, \mathcal{RC}, \mathcal{LE}, \mathcal{LC}\}$
11:       Update $S_x^{\mathcal{A}}$, $S_{x^2}^{\mathcal{A}}$, $S_y^{\mathcal{A}}$, $S_{y^2}^{\mathcal{A}}$, $S_{xy}^{\mathcal{A}}$ incrementally
12:       Compute $\rho(Q_i)$;
13:       $\Delta_{Q_i} = \lambda\Delta_{Q_i}^{E} + (1-\lambda)\Delta_{Q_i}^{\rho}$;
14:       if ( $\Delta_{Q_i} < best$ ) then
15:          $best = \Delta_{Q_i}$ ; $Q_p \leftarrow Q_i$;
16:       $\Delta_{TH} = \lambda\Delta_{Q_i}^{E}$;
17:       $\mathcal{Q}$.push($Q_i$);
18: return $Q_p$, $best$;
```

3.3. I/O-centric Algorithm

Intuitively, estimating the correlation of a given candidate query Q_i as an interval $C_I = [c_1 - c_u]$, such that the real correlation value of Q_i is guaranteed to be within this interval, enables an algorithm to prune Q_i based on a trivial check. Specifically, by knowing the best subsequence deviation found so far (i.e., best), C_I is substituted into Eq. 4 and the returned deviation is compared against the best solution. Then, Q_i is pruned when C_I returns higher deviation than the best, which results into saving I/O cost.

3.3.1. Correlation Interval (CI) :

We propose to utilize the relationship between the Pearson's correlation and Z-normalized Euclidean distance between any two time series *x, y* [21] to enable pruning unpromising candidate queries:

$$\rho(x,y) = 1 - \frac{\left(d_2(\hat{x},\hat{y})\right)^2}{2} \tag{5}$$

$$d_2(\hat{x},\hat{y}) = \sqrt{2 - 2\rho(x,y)} \tag{6}$$

Where $\hat{x}, \hat{y}$ are the *Z*-normalized versions of time series *x*, *y*. As the above equations show, the Pearson correlation value increases towards 1 when the normalized distance decreases, towards zero, and vice versa [5]. Hence, we can compute the equivalent of C_I as a normalized distance interval $d_I = [d_l - d_u]$, then substitute in the above equations to get C_I.

In order to find the normalized distance interval d_I of a current, unprobed candidate query Q_i, we use the overlapping section of an already probed query with Q_i, and put new values in place of the unknown ones. After that, the normalization is performed and the distance interval d_I is computed. Next, we explain a simple, easy to maintain technique which facilitates the computation of d_I.

Algorithm 4 Met-data Matrix $\mathcal{M}_x$

```
1: Input: Time series $x$
2: Return: 2x2 Meta-data matrix $\mathcal{M}_x$
3: for ( $x_i$, $i \in [1, 2, ..., n-1]$ ) do
4:    if ( $x_i \leq x_{i+1}$ ) then
5:       $f_{upwards} = |x_i - x_{i+1}|$;
6:       add $f_{upwards}$ to $\mathcal{M}_x$ if min/max;
7:    else
8:       $f_{downwards} = |x_i - x_{i+1}|$;
9:       add $f_{downwards}$ to $\mathcal{M}_x$ if min/max;
10: return $\mathcal{M}_x$;
```

3.3.2. Meta-Data

To find the distance interval d_I, new values have to be assumed in place of the unknown ones. Those new values are estimated as the extreme cases of any value in the time series. In particular, for a time series x, we keep a meta- data, small 2x2 matrix:

M_x	Minimum	Maximum
Upwards	..	..
Downwards	..	..

Table 2. M_x: 2x2 Meta-data matrix for time series x

This matrix M_x summarizes the extreme historical behavior of x's values. Namely: the upwards/downwards, minimum/maximum change between any two consecutive values of x. Therefore, the new value to be added to a subsequence with one missing value will be in the range of the four cases in M_x.

Note that this technique performs a similar functionality as any interpolation method (e.g., linear regression, curve fitting), however our goal here is to find the range that is guaranteed to have the real value within, rather than approximating that value. Algorithm 4 shows the computation steps of M_x for a given time series x. Next, we present our proposed I/O-centric algorithm which utilizes M_x to find d_I for pruning unpromising queries and reducing I/O cost.

3.3.3. Algorithm

Our I/O-centric algorithm (IOC) has the same steps as INC. However, differently than INC, IOC avoids probing the DBMS for each candidate query with guarantees on the solution accuracy. Hence, IOC only probes a candidate query Q_i if it has a chance to be the best solution.

Using an additional structure T to store each probed query for future usage, IOC checks if a candidate query Q_i happened to have no overlap with any query in T. If so, then a probe is made to DBMS. However, since the auxiliary function *Refine*() produces neighboring queries, there is more likely to be an overlapping query Q_v for each Q_i, except for the first ever Q_i.

The core of IOC are the lines 10-12 in Alg 5. In line 10, the found overlapping query Q_v is used to populate the unprobed query Q_i. Finding out the non-overlapping values though, requires probing the DBMS. However, IOC efficiently avoids this expensive operation, whenever possible, as explained next.

IOC exploits M_x and M_y to estimate what would be the new, non-overlapping values of Q_i. That is, for the two subsequences from x and y in Q_i, there would be a total of 16 possible versions of Q_i, according to M_x and M_y. For instance, one of those 16 versions can be the following query (see Figure 5): The new estimated value v_k+1 of x(y) is the minimum upwards of M_x(M_y) plus the previous value v_k of x (y).

From those 16 versions, the minimum and maximum distances are selected as the distance interval d_I, and then converted into a correlation interval using Eq. 5 (line 11).

Lemma 1. Given M_x and M_y If $d_2(\hat{x}, \hat{y})$ is the normalized distance between x and y, then there is a lower and upper bound distance $d_2'(\hat{x}, \hat{y}) \leq d_2(\hat{x}, \hat{y}) \leq d_2''(\hat{x}, \hat{y})$ such that the last values in x and y are replaced with updated values from M_x and M_y.

Proof. Let x_m and x_m be the real values. Also, let $x'_m \leq x_m$ and $y'_m \leq y_m$ be the updated values using M_x and M_y. We have:

$$d_2(\hat{x}, \hat{y}) = \sqrt{\Sigma_{i=1}^{m} (\hat{x_i} - \hat{y_i})^2}$$
$$= \sqrt{\Sigma_{i=1}^{m-1} (\hat{x_i} - \hat{y_i})^2 + (\hat{x_m} - \hat{y_m})^2}$$

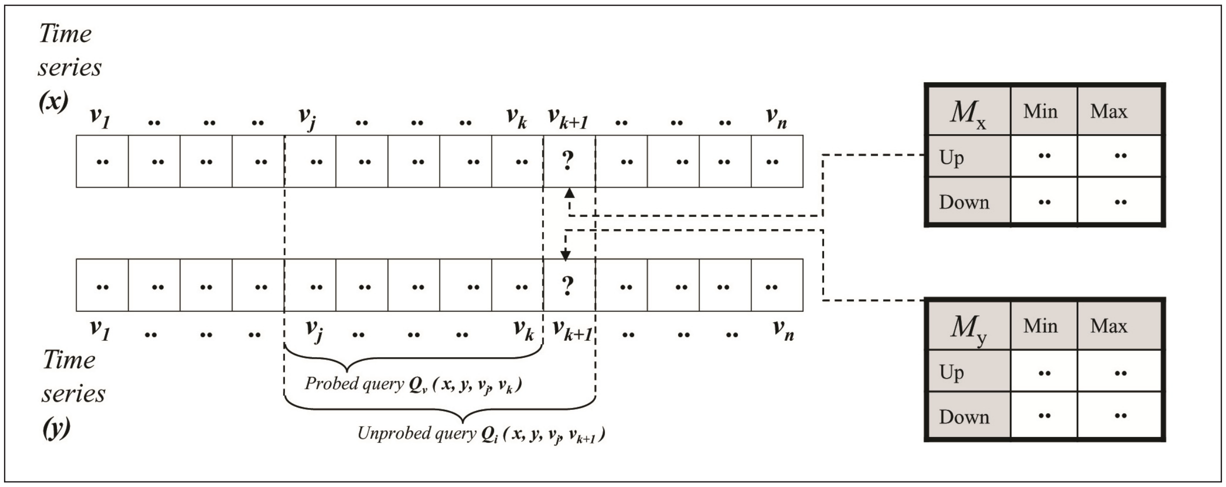

Time series (x)
v_1 .. .. .. v_j .. .. .. v_k v_{k+1} .. .. .. v_n
?
M_x
Min
Max
Up
Down
M_y
Time series (y)
Probed query Q_v (x, y, v_j, v_k)
Unprobed query Q_i (x, y, v_j, v_{k+1})

Figure 5. The new unknown values v_{k+1} in Q_i are estimated by M_x and M_y, while the rest are taken from the overlapping query Q_v

Replacing $\hat{x_m}$ and $\hat{y_m}$ with $\hat{x_m'}$ and $\hat{y_m'}$:

$$d_2'(\hat{x}, \hat{y}) = \sqrt{\Sigma_{i=1}^{m-1}(\hat{x_i} - \hat{y_i})^2 + (\hat{x_m'} - \hat{y_m'})^2}$$

Since $(\hat{x_m'} - \hat{y_m'})^2 \leq (\hat{x_m} - \hat{y_m})^2$, then it follows that the lower bound $d_2'(\hat{x}, \hat{y}) \leq d_2(\hat{x}, \hat{y})$. The upper bound proof is similar.

By experiments, we show the efficiency of IOC by comparing it to another version of itself that does not prune candidate queries based on the correlation interval.

4. Experiments Setup and Results

We conducted a set of experiments on real-world and synthetic datasets to evaluate our algorithms on a PC with Intel Core i7 @ 3.40 GHz, 16 GB of RAM. Algorithms were coded in Java. We report the results next.

Algorithm 5 I/O centric Algorithm (IOC)

```
1: Input: Input query $Q_I(x, y, s, e)$, preference weight $\lambda$
2: Return: $Q_p$, $best$;
3: $best$ = (1-$\lambda$)$\Delta^{\rho}_{Q_I}$; $Q_p \leftarrow Q_I$; $T = \phi$;
4: $\mathcal{Q}$.push($Q_I$);
5: while ( $\mathcal{Q} \neq \phi$ & $\Delta_{TH} \leq best$) do
6:    $Q_c \leftarrow$ Refine($\mathcal{Q}$.pop());
7:    for ( each $Q_i \in Q_c$ s.t. $Q_i$ not visited ) do
8:       Lookup $Q_v$ for $Q_i$ in $T$;
9:       if ($Q_v \neq \phi$) then
10:          Update $Q_i$ with $Q_v$ & $\mathcal{M}$;
11:          Compute $[d_l - d_u]$; convert to $[c_l - c_u]$;
12:          if ( $\Delta'_{Q_i} \leq best$ ) then
13:             Probe $Q_i$; add $Q_i$ to $T$;
14:             if ( $\Delta_{Q_i} \leq best$ ) then
15:                $best = \Delta_{Q_i}$ ; $Q_p \leftarrow Q_i$;
16:       else
17:          Probe $Q_i$; add $Q_i$ to $T$;
18:          if ( $\Delta_{Q_i} \leq best$ ) then
19:             $best = \Delta_{Q_i}$ ; $Q_p \leftarrow Q_i$;
20:       $\Delta_{TH} = \lambda\Delta^{E}_{Q_i}$;
21:       $\mathcal{Q}$.push($Q_i$);
22: return $Q_p$, $best$;
```

Parameter	Default	Range
Preference weight (λ)	0.5	0.0 – 1.0
Target correlation (tc)	-	0.0 – 1.0
Query/time series ratio	%20	%10 – %100

Table 3. Parameters of the experiments

4.1. Setup
Table 3 summarizes all parameters used throughout the experiments and the dataset settings.

Datasets: We have experimented with a real dataset and a synthetic one as well. The real dataset contains 61 time series. Each time series represents the historical daily closing price of a company listed in the US stocks market from 2/1/2009 to 31/12/2013, n = 1; 258≈10^3. A sample of this dataset is shown in Figure 1 for two companies: Microsoft Corporation (MSFT) and Cisco Systems (CSCO). This dataset was manually extracted from Yahoo! Finance and had to be merged and processed to be suitable for experimenting.

Also, we have experimented with a synthetically generated time series dataset using the Random Walk model [15].

There are a total of 10 time series in this dataset, with length n = 2; 000. The Random Walk model was set so that each step is chosen from a Normal distribution with $\mu = 0$ and $\sigma = 1$.

Performance Measures: In this paper, we proposed two algorithms and implemented another two for comparison. We compare the computational-centric algorithms against others using the CPU cost as an indicator for efficiency, while the I/O centric algorithm's performance is measured using the number of I/O calls made to the DBMS (I/O cost). The CPU cost of an algorithm is the number of oat point operations (FLOPs) the algorithm had to make to find the optimal solution. Specifically, those operations are the addition, subtraction, multiplication and division operations of numbers made to obtain the Pearson correlation.

As for effectiveness, we use the total deviation Δ defined in Eq. 4.

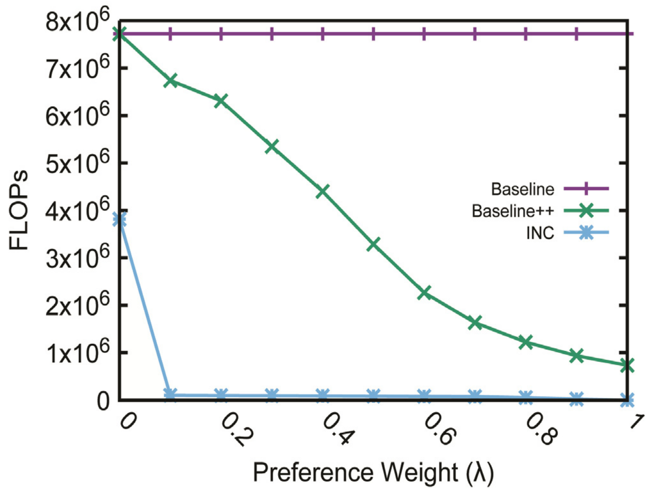
We averaged FLOPs, I/O calls and Δ for a workload of 10^3 input queries. Specifically, each item in the workload consists of 10 pairs (chosen randomly out of k) time series, 10 uniformly distributed time intervals [s, e] and 10 uniformly distributed target correlation ranges $[c_1 - c_u]$ where: $0 \leq c_1,\ c_u \leq 1$ and $c_1 \leq\ c_u$. Note that a pair of time series and a time interval construct an input query Q_I.

Because of the different optimization goals of the proposed algorithms, we first compare: Baseline, Baseline++ (limited), and Incremental (INC) against each other under different parameters, then later we compare (IOC) against a baseline of itself for some parameters as well.

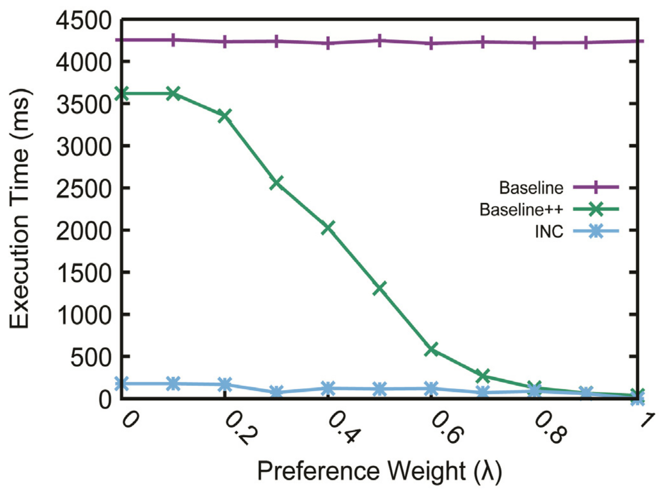
4.2. Results
4.2.1 Preference Weight (λ):
To show the impact of λ, we evaluated the algorithms with our workload while varying λ and averaged the result. Figures 6a and 6c show the average FLOPs and average deviation respectively.

All three algorithms achieve the same accuracy in terms of deviation. However, they exhibit different costs. When compared to the Baseline and Baseline++, INC manages to find the same solution as them but with almost half the cost when $\lambda = 0$. The reason is, INC keeps four sets of the cumulative arrays. One for each move. This enables INC to update the cumulative arrays by only adding or subtracting one value (one oat point operation) from the previously computed cumulative arrays, while Baseline and Baseline++ have to add and subtract two values (two oat point operations) from the previously computed cumulative arrays.

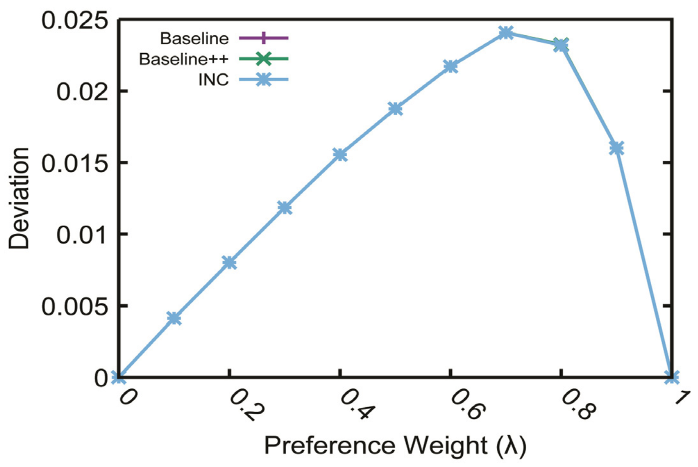
By increasing the preference weight, both Baseline++ and INC costs decrease by a considerable amount while Baseline stays constant. The reason is: increasing the weight of preference for Baseline++ and INC is orthogonal to increasing the tightness of the search space in Baseline++ and to early abandoning the search by the threshold Δ_{TA} in INC. Also, Figure 6c shows a tradeo between the conicting two objectives, which follows a semi-bell shape pattern.



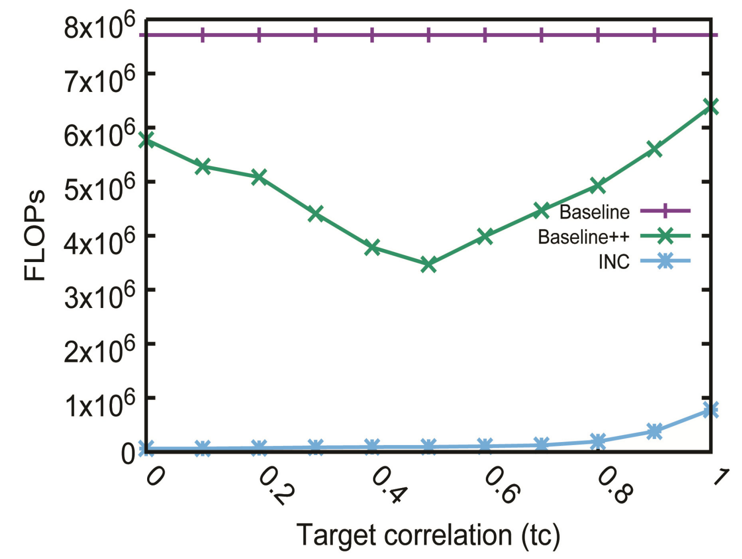
(a) Average FLOPs while varying λ



(b) Average execution time (ms) while varying λ



(c) Average deviation while varying λ



(d) Average FLOPs while varying tc with fixed $\lambda = 0.5$

Figure 6. Set of experiments on real dataset extracted from Yahoo! Finance

4.2.2 Target Correlation (tc)

To show the relationship between tc and FLOPs, we set $\lambda = 0.5$ and varied tc such that $c_l = c_u$. As Figure 6d shows, Baseline exhibit a constant performance regardless of the value tc, while Baseline++ is the most affected by tc. The closer tc is to 0.5 (which happens to be the average correlation among all time series in the dataset), the less work Baseline++ will have to do to achieve the optimal solution. In other words, Baseline++ will achieve tighter search space if tc is close from the input query's correlation $\rho(Q_l)$ since best in Alg. 2 line 3 will be very close from zero.

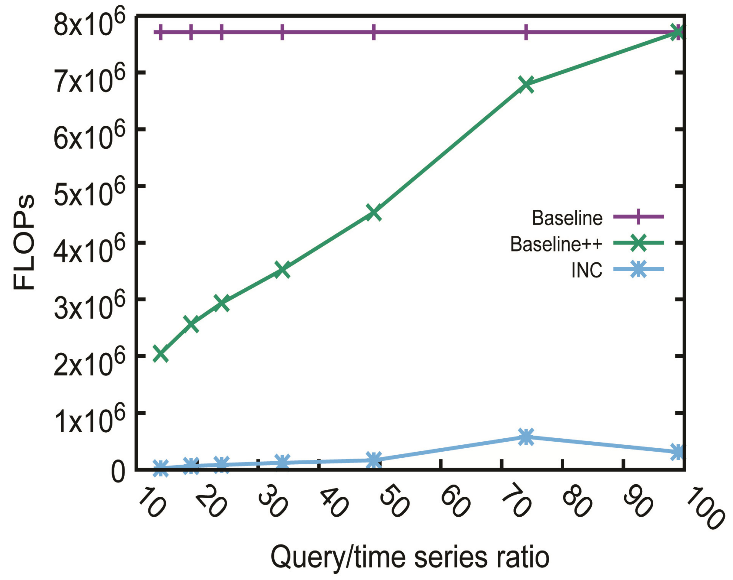


Figure 7. Average FLOPs while varying query/time series ratio

4.2.3. Query/Time Series Length Ratio
As mentioned previously, the query/time series length ratio has an effect on Baseline++, as shown in Figure 7, while INC's performance is relatively constant. Specifically, the more the query's length reaches the time series length, the less effective Baseline++ is in limiting the search space.

For instance, if the query's length is the same as the time series length, i.e., %100, then basically Baseline++ will consider the whole time series as its limited search space, and its performance will converge to Baseline's performance.

4.2.4 I/O-centric Algorithm
To demonstrate the efficiency of the correlation-interval based pruning technique of IOC (see Section 3.3), we created another version of itself but without this pruning technique (called IOC wo/). Using the synthetic dataset, Figures 8a and 8b report the average number of I/O calls while varying query/time series ration.

When users preferences are neglected, i.e., $\lambda = 0$, IOC wo/ performs no pruning at all, hence, the constant number of I/O calls for different ratios. On the other hand, IOC with pruning is able to prune some unnecessary queries. In particular, IOC w/ prunes a decent amount of candidate queries resulting into a maximum of %40 reduction in I/O cost when compared with IOC without pruning.

5. Related Works

There are two bodies of research that touch on the preference aware correlation-based query refinement problem: query refinement, and correlated time series search. Query refinement has been the focus of researchers for different reasons [3, 12, 20, 8, 16, 11]; generating queries with cardinality constraints to test databases engines, refining empty-result queries, and interactively refining failing queries to satisfy cardinality constraints. Focused on time series, this work has its own characteristics that differentiate it from those works. We can abbreviate the differences in twofold: the search space, and the objective.

Generally, query refinement techniques model the search space in a way to facilitate the application of their optimization techniques. For instance, TQGen [12] models the search space as a d-dimensional grid such that each cell represents a query. Then, to avoid an exhaustive search for all cells, a scoring function is applied first and only the top b cells are selected to explore. Another example is QRelX [20] that also models the search space as a d-dimensional grid. However differently than TQGen, it explores the cells in a way to enable computations reusing, resulting in low processing time and high latency.

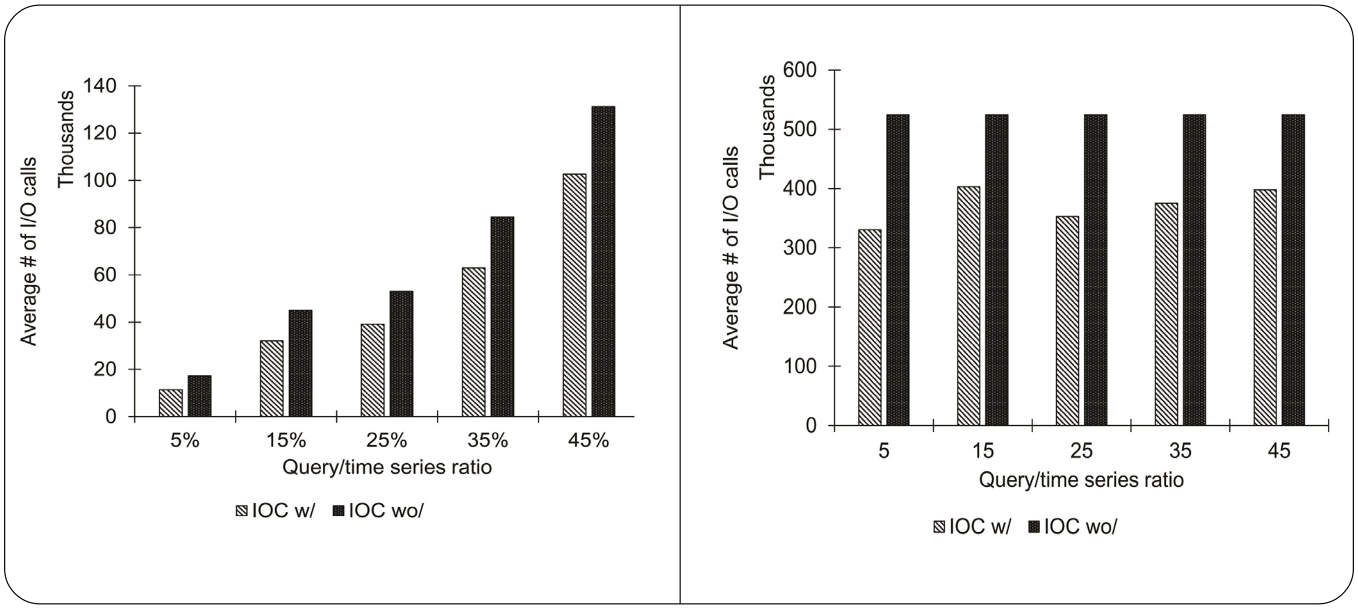


Figure 8. Comparison of IOC with correlation interval pruning (IOC w/) and without (IOC wo/) on the synthetic dataset

As for the objective, SKIP [10], Jocor [14] and our work share a common objective. The goal of SKIP is to find the longest subsequence of two time series that have a correlation higher than a threshold value , while Jocor aims to find the longest correlated subsequence such that the length is higher than a threshold value ml. However, differently than SKIP and Jocor, the RELATE problem we tackled in this paper considers users' preferences as an additional objective for the correlation-based subsequence search problem. Moreover, RELATE stretches the correlation constraint to be a range instead of a threshold value, as in SKIP and Jocor. Stretching the correlation constraint to be a range calls for new optimization techniques in order to nd the optimal solution efficiently.

6. Conclusions

Motivated by the prevalent need to support exploring the continuously growing time series databases, we proposed the Preference-aware Correlation-based Query Refinement problem (RELATE). By an example, we showed how important it is to search for time series based on target correlation thresholds while considering users preferences. Subsequently, we formally dened RELATE and showed how a simple extension to state-of-the-art solution is feasible, however there is a lot of optimization opportunities in terms of CPU and I/O costs, that can be utilized to support interactive exploration of time series.

Therefore, we proposed a computational centric algorithm (INC) which provides much more efficiency without sacrificing on the solution quality by utilizing the users preferences objective to prune unpromising candidate queries, and maintaining cumulative arrays to boost correlation computations. Further, we proposed an I/O centric algorithm (IOC) which optimizes the search problem from an I/O cost point-of-view by estimating correlation as a range for a given candidate query. In particular, IOC makes use of meta-data gathered from each time series to estimate any candidate query's correlation as a range, and avoids probing the DBMS if this candidate query is most likely to be not optimal, resulting in saving I/O cost.

References

[1] Albarrak, A. M., Sharaf, M. A. (2017). Efficient schemes for similarity-aware refinement of aggregation queries. World Wide Web, 20 (6) 237-1267, 2017.

[2] Bikakis, N., Sellis, T. K. (2016). Exploration and visualization in the web of big linked data: A survey of the state of the art. *In:* Proceedings of the Workshops of the EDBT/ICDT 2016 Joint Conference, 2016.

[3] Bruno, N., Chaudhuri, S., Thomas, D. (2006). Generating queries with cardinality constraints for DBMS testing. *IEEE Trans. Knowl. Data Eng*., 18 (12) 1721-1725.

[4] Buoncristiano, M., Mecca, G., Quintarelli, E., Roveri, M., Santoro, D., Tanca, L. (2015). Database challenges for exploratory computing. *SIGMOD Record,* 44 (2) 17{22, 2015.

[5] Guo, T., Sathe, S., Aberer, K. (2015). Fast distributed correlation discovery over streaming time-series data. *In:* Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, CIKM 2015, Melbourne, VIC, Australia, October 19 - 23, 2015, p. 1161-1170, 2015.

[6] Idreos, S., Papaemmanouil, O., Chaudhuri, S. (2015). Overview of data exploration techniques. *In:* Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015, p. 277-281, 2015.

[7] Jiang, L., Nandi, A. (2015). Snaptoquery: Providing interactive feedback during exploratory query specification. PVLDB, 8(11) 1250-1261, 2015.

[8] Kadlag, A., Wanjari, A. V., Freire, J., Haritsa, J. R. (2004). Supporting exploratory queries in databases. *In:* DASFAA, p. 594-605, 2004.

[9] Kersten, M. L., Idreos, S., Manegold, S., Liarou, E. (2011). The researcher's guide to the data deluge: Querying a scientific database in just a few seconds. PVLDB, 4(12)1474-1477, 2011.

[10] Li, Y., L., Yiu, H. U, M. L., Gong, Z. (2013). Discovering longest-lasting correlation in sequence databases. PVLDB, 6 (14) 1666-1677, 2013.

[11] Mishra, C., Koudas, N. (2009). Interactive query refinement. *In:* EDBT 2009, 12th International Conference on Extending Database Technology, Saint Petersburg, Russia, March 24-26, 2009, Proceedings, p. 862-873, 2009.

[12] Mishra, C., Koudas, N., Zuzarte, C. (2008). Generating targeted queries for database testing. *In:* Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008, p. 499-510, 2008.

[13] Mottin, D., Marascu, A., Roy, S. B., Das, G., Palpanas, T., Velegrakis,Y. (2014). IQR: an interactive query relaxation system for the empty-answer problem. *In:* International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014, p. 1095-1098, 2014.

[14] Mueen, A., Hamooni, H., Estrada, T. (2014). Time series join on subsequence correlation. *In:* 2014 IEEE International Conference on Data Mining, ICDM 2014, Shenzhen, China, December 14-17, 2014, p. 450{459, 2014.

[15] Mueen, A., Nath, S., Liu, J. (2010). Fast approximate correlation for massive time-series data. *In:* Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010, p. 171-182, 2010.

[16] Muslea, I. (2004). Machine learning for online query relaxation. In KDD, p. 246-255, 2004.

[17] Sakurai, Y., Papadimitriou, S., Faloutsos, C. (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, p. 599-610, 2005.

[18] Tran, Q. T., Chan, C. (2010). How to conquer why-not questions. *In:* Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010, p. 15-26, 2010.

[19] Vartak, M., Raghavan, V., Rudensteiner, E., Madden, S. (2016). Refinement driven processing of aggregation constrained queries. In Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016., p. 101-112, 2016.

[20] Vartak, M.,Raghavan, V., Rundensteiner, E. A. (2010). Qrelx: generating meaningful queries that provide cardinality assurance. *In:* Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010, p. 1215-1218, 2010.

[21] Zhu, Y., and Shasha, D. (2002). Statstream: Statistical monitoring of thousands of data streams in real time. *In:* VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China, p. 358-369, 2002.