Mach Learn (2011) 83: 103–131
DOI 10.1007/s10994-010-5204-9

Checkpoint evolution for volatile correlation computing

Wenjun Zhou · Hui Xiong

Received: 10 September 2009 / Revised: 23 May 2010 / Accepted: 27 June 2010 /
Published online: 24 July 2010

Abstract Given a set of data objects, the problem of correlation computing is concerned with efficient identification of strongly-related ones. Existing studies have been mainly focused on static data. However, as observed in many real-world scenarios, input data are often dynamic and analytical results have to be continually updated. Therefore, there is the critical need to develop a dynamic solution for volatile correlation computing. To this end, we develop a checkpoint scheme, which can help us capture dynamic correlation values by establishing an evolving computation buffer. In this paper, we first provide a theoretical analysis of the properties of the volatile correlation, and derive a tight upper bound. Such tight and evolving upper bound is used to identify a small list of candidate pairs, which are maintained as new transactions are added into the database. Once the total number of new transactions goes beyond the buffer size, the upper bound is re-computed according to the next checkpoint, and a new list of candidate pairs is identified. Based on such a scheme, a new algorithm named CHECK-POINT+ has been designed. Experimental results on real-world data sets show that CHECK-POINT+ can significantly reduce the computation cost in dynamic data environments, and has the advantage of compacting the use of memory space.

Keywords Pearson's correlation coefficient · Correlation coefficient · Volatile correlation computing · Checkpoint

Editor: Johannes Fürnkranz.

A previous paper by the same authors was Runner-up for the Best Student Paper Award at the 2008 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. The research described in this paper builds upon and extends the work reported in the KDD-2008 paper.

W. Zhou · H. Xiong (✉)
MSIS Department, Rutgers University, 1 Washington Park, Newark, NJ 07102, USA
e-mail: hxiong@rutgers.edu

W. Zhou
e-mail: wjzhou@pegasus.rutgers.edu

1 Introduction

Recent years have witnessed increased interest in computing strongly related data objects (e.g., strongly correlated item pairs, as measured by Pearson's correlation coefficient). Many important applications in science and business (e.g. Alexander 2001; Cohen et al. 2002; Kuo et al. 2002) depend on efficient and effective correlation computing techniques to discover relationships within large collections of data.

Despite the development of correlation computing techniques (e.g. Brin et al. 1997; Jermaine 2001; DuMouchel and Pregibon 2001; Jermaine 2003; Ilyas et al. 2004; Xiong et al. 2004, 2006), researchers and practitioners are still facing increasing challenges to measure associations among data objects produced by emerging data-intensive applications, particularly when the data are dynamic and analytical results need to be continually updated. Indeed, with such large and growing data sets, research efforts are needed to develop a dynamic solution for volatile correlation computing. To that end, in this paper we provide a pilot study of dynamically finding all strongly related item pairs, whose correlation values are above a user-specified minimum threshold, as new data are constantly being collected.

As motivating examples, let us consider the following potential application scenarios. First, consider an e-commerce Web site that would like to promote sales by making recommendations to customers. In order to automate this process, the computerized system can recommend items most highly correlated to those being purchased, according to past transactions. As new orders are being placed, the recommendations should be updated automatically, and reflect recent interests in a timely manner. In this case, the underlying computation would involve finding the strongly correlated item pairs in a real-time fashion. A second example can be found in automatic stock picking. In order to monitor stocks with correlated price movements, a portfolio manager might be interested in knowing highly correlated stocks, whose prices tend to move in the same direction (either going up or going down). Despite the large number of stocks on the market, and the number of days with price quotes, the portfolio manager may want to maintain the up-to-date list of strong pairs as his decision support. Both the above application scenarios require efficient computation of strongly correlated pairs in a dynamic fashion.

A straightforward solution is to recompute the correlations of all item pairs every time when new data become available. However, for large data sets, this approach is not practical, particularly if the application needs the results in a timely fashion. An alternative method is to use more space to save time. Along this line, we present a SAVE-ALL algorithm, which saves the intermediate results for all item pairs. When new transactions are added into the database, SAVE-ALL only updates the stored values corresponding to each item pair, and computes the correlation query results with the intermediate values. Obviously, the SAVE-ALL method compromises space for time. If the number of items in the data set becomes considerably large, the number of pairs grow even larger, to the extent that it is impossible to save the intermediate computing results of all item pairs in the memory space. This motivates our interest in volatile correlation computing.

In our preliminary work (Zhou and Xiong 2008), we proposed a CHECK-POINT algorithm that makes a time-space tradeoff and can efficiently incorporate new transactions for correlation computing as they become available. In CHECK-POINT, we set a checkpoint to establish a computation buffer, which can help us determine a correlation upper bound. This checkpoint bound can be exploited to identify a list of candidate pairs, whose frequencies are maintained and correlations are computed, as new transactions are being added into the database. However, if the total number of new transactions exceeds the buffer size, a new upper bound is computed according to the next checkpoint and a new list of candidate pairs is identified.

The rationale behind CHECK-POINT is that, if the number of new transactions is much smaller than the total number of transactions in the database, the correlation coefficients of most item pairs will not change substantially. In other words, we only need to establish a very short list of candidate pairs at the checkpoint and maintain this candidate list in the memory as new transactions are added into the database. Unlike SAVE-ALL, CHECK-POINT only maintains the intermediate computing results of a very small portion of the item pairs. This can greatly compact the use of the memory space, using slightly more time.

In this paper, we derive a tight upper bound for the evolving correlation. The tight bound is exploited to identify a more compact candidate list than CHECK-POINT, so that better correlation computing performances can be achieved. Also, we identify the local monotone property of this new upper bound. Such property can be used for searching the optimal points effectively. In addition, based on this new upper bound, we exploit the local monotone property and design a CHECK-POINT+ algorithm for volatile correlation computing.

As demonstrated by our experimental results on several real-world data sets, CHECK-POINT+ has a much better computational performance than CHECK-POINT. Both CHECK-POINT and CHECK-POINT+ can significantly reduce the computational cost compared to existing correlation computing benchmark algorithms, e.g. TAPER, in dynamic data environments. Also, compared to CHECK-POINT, CHECK-POINT+ is less sensitive to the change of parameters. Moreover, we observe that there is a trade-off between the use of time and space by setting different checkpoint values. Indeed, the size of the candidate list decreases with the increase of the checkpoint density. In contrast, the average computational savings is reduced with the increase of the checkpoint density. Finally, our experimental results show that CHECK-POINT+, as compared to SAVE-ALL and CHECK-POINT, can greatly reduce the use of memory space.

The rest of this paper is organized as follows. Section 2 presents some basic concepts and formulates the problem. In Sect. 3, we discuss the related work. Section 4 provides a checkpoint view of dynamic correlation computing. In Sect. 5, we derive an evolving upper bound of ϕ correlation coefficient. Section 6 describes the CHECK-POINT+ algorithm. In Sect. 7, we show the experimental results on real-world data. Finally, Sect. 8 concludes the work.

2 Volatile correlation computing

In this section, we first provide the computation formula of ϕ correlation coefficient between any pair of items. Then, we formulate the problem of volatile correlation computing.

2.1 ϕ correlation coefficient

The ϕ correlation coefficient is the computation form of the Pearson's correlation coefficient for binary variables (Pearson 1920; Reynolds 1977). Suppose that there are N transactions in the database, each of which consists of a list of unique items. For any two items a and b, we can produce a 2×2 contingency table like Table 1.

In Table 1, N_a is the number of transactions that contain item a, N_b is the number of those containing item b, and N_{ab} is the number of those containing both items. Then the ϕ correlation coefficient between items a and b can be computed as

$$\phi_{ab} = \frac{NN_{ab} - N_a N_b}{\sqrt{N_a(N - N_a)N_b(N - N_b)}}, \tag{1}$$

as derived in Xiong et al. (2004).

Table 1 The 2×2 contingency table for items a and b

		b		Row total
		1	0	
a	1	N_{ab}	$N_{a\bar{b}}$	N_a
	0	$N_{\bar{a}b}$	$N_{\bar{a}\bar{b}}$	$N_{\bar{a}}$
Column total		N_b	$N_{\bar{b}}$	N

2.2 Problem formulation

Here, we introduce the problem formulation. Let $\mathcal{D}$ be a transaction database, which has M items and N transactions. In this data set, a common task of correlation computing is to find all item pairs whose correlation coefficients are above a user-specified threshold θ. This is known as the all-strong-pairs (ASP) correlation query problem (Xiong et al. 2004). In this paper, we investigate the ASP correlation query problem in dynamic data environments.

Specifically, every time when S new transactions is added into the original database $\mathcal{D}$, we want to have the dynamically updated results from the ASP correlation query. In other words, this ASP correlation query can be a frequent task in dynamic data environments. In this study, our goal is to develop a dynamic, practical, and computation-efficient solution to this ASP correlation query problem.

3 Related work

Association analysis has been a major topic in the field of data mining. Traditional association-rule mining algorithms (Agrawal et al. 1993; Bayardo and Agrawal 1999) are found to be yielding many spurious patterns (Brin et al. 1997; Xiong et al. 2006, 2008). As a result, in recent years, many statistical correlation measures, such as χ^2 statistics (Brin et al. 1997; DuMouchel and Pregibon 2001; Jermaine 2001, 2003), Pearson's correlation coefficients (Xiong et al. 2004, 2006, 2008; Zhou and Xiong 2008), rank-based correlation coefficients (Melucci 2007; Yilmaz et al. 2008), and mutual information (Ke et al. 2006, 2007) have been considered in the setting of large-scale association analysis.

Specifically, due to its flexibility in measuring the association among multiple items, the χ^2 statistic has been proposed to analyze market-basket data (Brin et al. 1997). However, χ^2 does not possess an upward closure property for efficient computation (DuMouchel and Pregibon 2001). Also, Jermaine (2001) investigated the implication of incorporating χ^2-based queries into data cube computations. He showed that finding subcubes that satisfy statistical tests such as χ^2 are inherently NP-hard, but may be made more tractable using approximation schemes. Finally, Jermaine (2003) presented an iterative procedure for correlation analysis by dismissing part of the database based on human feedback.

On the other hand, the upper bound of ϕ, which is the binary form of Pearson's correlation coefficient (Pearson 1920; Reynolds 1977), has been found to have an anti-monotone property (Xiong et al. 2004, 2006). Such property has helped finding all strongly related pairs efficiently, especially for data with Zipf rank-support distributions.

Pruning techniques using upper bounds have been effectively used on both ϕ (Xiong et al. 2004, 2006) and χ^2 (Morishita and Sese 2000). However, previous work does not consider dynamic data environments. In our preliminary work (Zhou and Xiong 2008), we proposed a CHECK-POINT algorithm that makes a time-space tradeoff for dynamic correlation computation. The idea was to use an upper bound of the evolving ϕ correlation by

setting a checkpoint for future transactions. In this paper, we improve the CHECK-POINT algorithm by deriving a tighter upper bound, which provides better pruning effect and can lead to better computational performances.

4 A checkpoint view

In this section, we introduce the checkpoint principle. Along this line, we provide some intuitive understandings of the checkpoint framework.

In general, there are two basic ways for developing incremental solutions for frequent and dynamic ASP correlation queries. First, the most straight-forward way is to recompute the correlation values for all the item pairs every time new data sets of transactions become available. In this case, we can use an efficient static ASP correlation query algorithm, such as TAPER (Xiong et al. 2004), for each run of query. However, for very large data sets, this approach is infeasible if data updates are very frequent and the results are needed in a timely manner. The second way is to use more space in order to save time (Bentley 2000). Specifically, we can save the support of each item pair and update the values every time new data are added into the database. In this way, once all the intermediate computing results are saved, the ASP correlation queries can be done very efficiently, but the memory requirement is very high. For instance, let us consider a database of 10^6 items, which may represent the collection of books available at an e-commerce Web site. There are $\binom{10^6}{2} \approx 0.5 \times 10^{12}$ possible item pairs, which need a huge amount of memory space to store intermediate computing results. In practice, this memory requirement cannot be satisfied for data sets with a large number of items.

The checkpoint-based method can be considered as an answer in between the above two solutions. Specifically, instead of saving the intermediate computing results for all item pairs, we propose to save them for only selected item pairs. Aiming for a tradeoff between time and space, we use a checkpoint to establish a computation buffer, which can help us determine an evolving correlation upper bound.

The basic process is illustrated in Fig. 1. At a checkpoint, assuming that there are n existing transactions in the database, and we know that Δ ($\Delta \ll n$) new transactions will be added before the next checkpoint, then we can develop an upper bound for all the item pairs on the $(n + \Delta)$ transactions. This upper bound has taken the newly added transactions into consideration. Therefore, based on this upper bound, we can establish a list of candidate item pairs whose upper bounds are greater than or equal to the threshold θ. This list of candidate item pairs can be treated as a computation buffer for ASP correlation queries. While new transactions can be added into the buffer dynamically, we only need to maintain

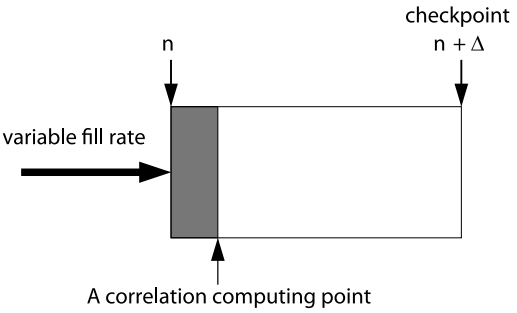


Fig. 1 An illustration of the checkpoint process

the intermediate results for item pairs in this candidate list as long as the cumulative number of new transactions is less than Δ.

The reason that the candidate list can remain unchanged (as long as the cumulative number of new transactions is less than Δ) is as follows. With a checkpoint at $(n + \Delta)$, we identify upper bounds for all item pairs for all n known transactions and Δ unknown transactions. In other words, these upper bounds are the maximum possible values they can achieve no matter what kind of Δ transactions have been added into the database. Then, if the cumulative number of new transactions is less than Δ, the upper bounds for all the item pairs in $(n + \Delta)$ transactions will remain unchanged. Therefore, the candidate list will also remain unchanged. We call this the checkpoint principle.

Once the cumulative number of new transactions is greater than Δ, we need to set a new checkpoint at $(n + 2\Delta)$. This iterative process will form an incremental solution for the dynamic ASP query problem. The rationale behind the checkpoint principle is that a small number of new transactions will not have significant impact on the correlation coefficients of most item pairs in the database if the total number of transactions is very large.

5 Evolving upper bound of ϕ correlation coefficient

In this section, we introduce an evolving upper bound of ϕ correlation coefficient. This upper bound is fundamental to the design of an efficient volatile correlation computing algorithm. We first start with the introduction of mathematical notations, which will be used throughout the paper. Then, we present a loose upper bound derived in our preliminary work (Zhou and Xiong 2008). Finally, we develop a tight upper bound based on our new understanding on the computational properties of the evolving correlation.

5.1 Mathematical notations

Here, we first present some basic mathematical notations used in this paper. Table 2 lists the mathematical notations to be used throughout this paper. Suppose that at a certain point of time, we have a database of n transactions, called the *original database*. Also, assume that at a later time, we will have collected another Δ new transactions, which we call the *incremental data*. By appending the new transactions to the original database, we will have a *combined database*, which has $(n + \Delta)$ transactions in total. Please note that n is always a known constant, and Δ is a tunable parameter.

For any item a, we denote its frequency in the original database as n_a, meaning that n_a of the n transactions contain item a. Therefore, the range of n_a will be $\{0, 1, 2, \ldots, n\}$. Also, for any item pair $\{a, b\}$, we denote its frequency as n_{ab}, meaning that n_{ab} of the original transactions contain both items a and b. We have $0 \leq n_{ab} \leq \min\{n_a, n_b\}$. These numbers and their relationships have counterparts in the incremental and the combined databases as

Table 2 Basic mathematical notations

Terms	Original	Incremental	Combined
Number of transactions	n	Δ	$N = n + \Delta$
Frequency of item a	n_a	Δ_a	$N_a = n_a + \Delta_a$
Frequency of item b	n_b	Δ_b	$N_b = n_b + \Delta_b$
Frequency of item pair $\{a, b\}$	n_{ab}	Δ_{ab}	$N_{ab} = n_{ab} + \Delta_{ab}$

well, as listed in Table 2. To simplify subsequent discussions, let $N = n + \Delta$, $N_a = n_a + \Delta_a$, $N_b = n_b + \Delta_b$, and $N_{ab} = n_{ab} + \Delta_{ab}$. Then the correlation of $\{a, b\}$ in the combined database follows directly from (1).

For the problem of volatile correlation computing, the frequency of any item or pair in the original database can be counted directly, so n_a, n_b and n_{ab} are known. However, since the Δ new transactions are to be collected in the future, the numbers in the incremental database, Δ_a, Δ_b and Δ_{ab}, are unknown.

5.2 A loose evolving upper bound

In our preliminary work (Zhou and Xiong 2008), a loose evolving upper bound for ϕ_{ab} has been derived. When separating the computation formula of ϕ_{ab} in (1) into three parts, we have

$$\phi_{ab} = \frac{W_{ab}}{v(N_a)v(N_b)},$$

where $W_{ab} = NN_{ab} - N_aN_b$ and

$$v(x) = \sqrt{x(N-x)}. \tag{2}$$

Then, we have

$$\phi_{ab} \le \frac{\max W_{ab}}{\min v(N_a) \min v(N_b)},$$

and the right hand side (RHS) is a loose upper bound for ϕ_{ab}. It has been derived in Zhou and Xiong (2008) that

$$\max W_{ab} = \begin{cases} w(0), & \text{if } c_0 \le 0; \\ w(c_0), & \text{if } 0 < c_0 \le \Delta; \\ w(\Delta), & \text{if } c_0 > \Delta; \end{cases}$$

where

$$c_0 = \frac{N - n_a - n_b}{2}, \tag{3}$$

$$w(t) = N(n_{ab} + t) - (n_a + t)(n_b + t); \tag{4}$$

and that $\min v(N_x) = \min\{v(n_x), v(n_x + \Delta)\}, \forall x \in \{a, b\}$.

Although $\max W_{ab}$, $\min v(N_a)$, and $\min v(N_b)$ may be reached individually, they may not be reached simultaneously. In other words, the upper bound on the RHS of (3) is rather loose. Note that, when the upper bound is larger than 1, it will not be helpful, since we know by definition that $\phi_{ab} \le 1$. This motivates us to look for a tight upper bound to achieve better pruning performances.

5.3 Search space analysis

For any item pair $\{a, b\}$, after the first n transactions, n_a, n_b and n_{ab} are known. In order to find a tight upper bound of the evolving ϕ_{ab} by the next checkpoint, we would like to find the maximal possible value of ϕ_{ab} by taking into account all possibilities within the next Δ new transactions. In this subsection, we analyze the range of (N_a, N_b), from which we will search for the optimal point.

Lemma 1 (Overall Search Space) *The search space of* (N_a, N_b), *on which* ϕ_{ab} *is defined*, *is* $R_{ab} = R_a \times R_b$, *where*

$$R_x = [n_x, n_x + \Delta] \cap \mathbb{Z} - \{0, N\}, \quad \forall x \in \{a, b\}, \tag{5}$$

and $\mathbb{Z}$ *denotes the set of all integers*.

Proof By definition, we have $N_a = n_a + \Delta_a$, where n_a is a known integer and Δ_a may take any value in $\{0, 1, 2, \ldots, \Delta\}$. So the range of N_a will be $\{n_a, n_a + 1, n_a + 2, \ldots, n_a + \Delta\} = [n_a, n_a + \Delta] \cap \mathbb{Z}$. However, according to the formula of ϕ_{ab} in (1), it is naturally required that $N_a \neq 0$ and $N_a \neq N$, otherwise the denominator of ϕ_{ab} will be zero, making ϕ_{ab} not meaningful. The same situation applies to N_b. Therefore, we exclude those special points from the search space. □

A graphical representation of R_{ab} can be shown in Fig. 2. In the coordinate system, if we denote the x-axis as N_a, y-axis as N_b, then the point (n_a, n_b) represents the current condition that so far we have n_a transactions containing item a and n_b transactions containing item b, and the shaded region represents the range of (N_a, N_b) within the next Δ new transactions. For any item pair $\{a, b\}$, the range R_{ab} must lie within $(0, N) \times (0, N)$, the boundary of which is shown with dashed lines, since we exclude the cases that $N_a = 0$, $N_b = 0$, $N_a = N$, or $N_b = N$.

Without loss of generality, in the following, we assume that $0 < N_a < N$ and $0 < N_b < N$ are always true, since the boundary cases seldom happen and are easy to exclude.

Lemma 2 (General Upper Bound) *The upper bound of* ϕ_{ab} *can be written as*

$$\text{upper}(\phi_{ab}) = \max \begin{cases} \max_{(N_a, N_b) \in R_{ab}^-} f(N_a, N_b; N(n_a - n_{ab})); \\ \max_{(N_a, N_b) \in R_{ab}^+} f(N_b, N_a; N(n_b - n_{ab})). \end{cases} \tag{6}$$

where

$$R_{ab}^- = R_{ab} \cap \{(N_a, N_b) : N_a - N_b \leq n_a - n_b\}; \tag{7}$$

$$R_{ab}^+ = R_{ab} \cap \{(N_a, N_b) : N_a - N_b \geq n_a - n_b\}; \tag{8}$$

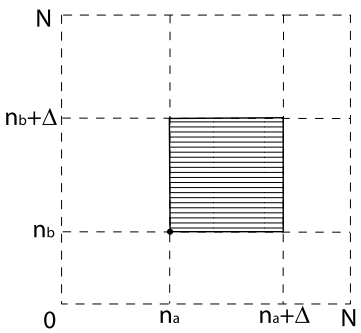


Fig. 2 The range of (N_a, N_b), $R_{ab} = R_a \times R_b$, as illustrated with the *shaded area*

and

$$f(x, y; c) = \frac{x(N - y) - c}{v(x)v(y)}. \tag{9}$$

with $v(x)$ *defined in* (2).

Proof According to the formula of ϕ_{ab} in (1),

$$\begin{aligned}
\phi_{ab} &\leq \frac{N(n_{ab} + \min\{\Delta_a, \Delta_b\}) - N_a N_b}{v(N_a)v(N_b)} \\
&= \begin{cases} \frac{N_a(N-N_b)-N(n_a-n_{ab})}{v(N_a)v(N_b)}, & \text{if } N_a - n_a \leq N_b - n_b \\ \frac{N_b(N-N_a)-N(n_b-n_{ab})}{v(N_a)v(N_b)}, & \text{if } N_a - n_a \geq N_b - n_b \end{cases} \\
&= \begin{cases} f(N_a, N_b; N(n_a - n_{ab})), & \text{if } N_a - N_b \leq n_a - n_b; \\ f(N_b, N_a; N(n_b - n_{ab})), & \text{if } N_a - N_b \geq n_a - n_b, \end{cases}
\end{aligned}$$

where $f(x, y; c)$ is the function given in (9).

As shown in Fig. 3, the line $N_b - N_a = n_b - n_a$ cuts the square region of R_{ab} into two isosceles right triangles, R^-_{ab} and R^+_{ab}. For any point $(N_a, N_b) \in R^-_{ab}$, $N_a - N_b \leq n_a - n_b$ (7). And for any point $(N_a, N_b) \in R^+_{ab}$, $N_a - N_b \geq n_a - n_b$ (8). Obviously, the maximum value of ϕ_{ab} on R_{ab} will be the larger of that on R^-_{ab} and R^+_{ab}. So the RHS of (6) will be an upper bound for ϕ_{ab}. □

In the following, we look for the optimal value (i.e. maximum value) of ϕ_{ab} on R^-_{ab} and R^+_{ab}, respectively. We find that in both cases, the search space R^-_{ab}, or R^+_{ab}, reduces to all integer points on the line segment

$$y = x - n_a + n_b, x \in [n_a, n_a + \Delta]; \tag{10}$$

while all other points need not to be considered.

Lemma 3 (Reduced Search Space in R^-_{ab}) *For any point* $(x, y) \in R^-_{ab}$, *it is always true that* $f(x, y; c) \leq f(x, x - n_a + n_b; c)$, *where* $c = N(n_a - n_{ab})$.

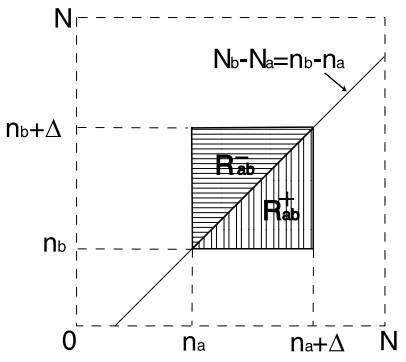


Fig. 3 The range of (N_a, N_b) in two parts, $R_{ab} = R^-_{ab} \cup R^+_{ab}$

Proof For any $(x, y) \in R_{ab}^{-}$, we have $c - Nx \leq N(n_a - n_{ab}) - Nn_a = -Nn_{ab} \leq 0$. Taking partial derivatives of $f(x, y; c)$ with respect to y (see Appendix A), we have

$$\frac{\partial f(x, y; c)}{\partial y} = \frac{(N - y)(c - Nx) - cy}{2v(x)v^3(y)} \leq 0. \tag{11}$$

So fixing x, $f(x, y; c)$ increases as y decreases. Since $y \geq x - n_a + n_b$, we have $f(x, y; c) \leq f(x, x - n_a + n_b; c)$. □

According to (7), for any point $(x, y) \in R_{ab}^{-}$, we have $y \geq x - n_a + n_b$. As shown in Lemma 3, we can always find a point (x, y') on (10), where $y' = x - n_a + n_b \leq y$, such that $f(x, y'; c) \geq f(x, y; c)$. In other words, the search space in R_{ab}^{-} reduces to integer points on the line segment in (10), and there is no need to check any other point in R_{ab}^{-}.

Lemma 4 (Reduced Search Space in R_{ab}^{+}) *For any point $(x, y) \in R_{ab}^{+}$, it is always true that $f(y, x; c) \leq f(x - n_a + n_b, x; c)$, where $c = N(n_b - n_{ab})$.*

Proof For any $(x, y) \in R_{ab}^{+}$, we have $N(N - x) - c \geq N(N - n_a - \Delta) - N(n_b - n_{ab}) \geq N(n - n_a - n_b + n_{ab}) \geq 0$. Taking partial derivatives of $f(y, x; c)$ with respect to y (see Appendix B), we have

$$\frac{\partial f(y, x; c)}{\partial y} = \frac{[N(N - x) - c]y + c(N - y)}{2v(x)v^3(y)} \geq 0. \tag{12}$$

So fixing x, $f(y, x; c)$ increases as y increases. Since $y \leq x - n_a + n_b$, we have $f(y, x; c) \leq f(x - n_a + n_b, x; c)$. □

According to (8), for any point $(x, y) \in R_{ab}^{+}$, we have $y \leq x - n_a + n_b$. As shown in Lemma 4, we can always find a point (x, y') on (10), where $y' = x - n_a + n_b \geq y$, such that $f(y', x; c) \geq f(y, x; c)$. In other words, the search space in R_{ab}^{+} reduces to integer points on the line segment in (10), and there is no need to check any other point in R_{ab}^{+}.

Lemma 5 (Upper Bound in Reduced Search Space) *The general upper bound in Lemma 2 can be simplified into*

$$\text{upper}(\phi_{ab}) = \max_{t \in \mathcal{T}} \frac{w(t)}{v(t + n_a)v(t + n_b)}. \tag{13}$$

where $v(x)$ and $w(t)$ are given in (2) and (4), respectively, and $\mathcal{T} = \{0, 1, \ldots, \Delta\}$.

Proof As proved in Lemmas 3 and 4, the optimal points within R_{ab} have to be somewhere on the line segment in (10). That is, when $N_b - n_b = N_a - n_a$. Let $t = N_b - n_b = N_a - n_a$, then

$$\begin{aligned} f(N_a, N_b; N(n_a - n_{ab})) &= \frac{(t + n_a)(N - t - n_b) - N(n_a - n_{ab})}{v(t + n_a)v(t + n_b)} \\ &= \frac{N(t + n_{ab}) - (t + n_a)(t + n_b)}{v(t + n_a)v(t + n_b)} \end{aligned} \tag{14}$$

$$f(N_b, N_a; N(n_b - n_{ab})) = \frac{(t+n_b)(N-t-n_a) - N(n_b - n_{ab})}{v(t+n_a)v(t+n_b)}$$
$$= \frac{N(t+n_{ab}) - (t+n_a)(t+n_b)}{v(t+n_a)v(t+n_b)}. \quad (15)$$

Notice that the RHS of (14) is exactly the same with the RHS of (15). So the upper bound in (13) is equivalent to (6). □

For the ease of writing, let

$$f(t) = \frac{w(t)}{v(t+n_a)v(t+n_b)}. \quad (16)$$

Then, based on Lemma 5, we have upper$(\phi_{ab}) = \max_{t\in\mathcal{T}} f(t)$. Now, all we need is to find max $f(t)$ on $\mathcal{T} = \{0, 1, \ldots, \Delta\}$.

5.4 Local monotonicity

In order to study the monotonicity of $f(t)$, we take the first derivative $f'(t)$, as derived in Appendix C. Note that although the range of t is a set of integers, when studying monotonicity, we treat $f(t)$ as a continuous function of t unless otherwise noted. Also, for the ease of writing, we always assume that values where $f(t)$ is undefined are excluded from the ranges we consider. We have

$$f'(t) = \frac{2(c_0 - t)[\alpha s(t) + \beta]}{v^3(t+n_a)v^3(t+n_b)}, \quad (17)$$

where

$$s(t) = t^2 - 2c_0 t + n_a n_b; \quad (18)$$
$$\alpha = N\left[n_{ab} - \frac{n_a + n_b}{2}\right] \leq 0; \quad (19)$$
$$\beta = N^2\left[n_a n_b - \frac{n_{ab}(n_a + n_b)}{2}\right]. \quad (20)$$

In the following, we analyze the monotonicity of $f(t)$ based on the sign of $f'(t)$. It has been found that depending on each pair, the monotonicity of $f(t)$ has up to three cases, as discussed in Lemma 6 through Lemma 8.

Lemma 6 (Monotonicity, Case I) *If $n_{ab} = n_a = n_b$, then $f(t)$ will be constant on $\mathcal{T}$.*

Proof When $n_a = n_b = n_{ab}$, from (19) and (20) we have $\alpha = 0$ and $\beta = 0$. In this case, (17) reduces to $f'(t) = 0$, so $f(t)$ is constant on $\mathcal{T}$. □

Whenever $n_a = n_b = n_{ab}$ does not hold, we have $\alpha < 0$. So (17) can be written as

$$f'(t) = \frac{-2\alpha(t - c_0)(t^2 - 2c_0 t + \gamma)}{v^3(t+n_a)v^3(t+n_b)}, \quad (21)$$

where

$$\gamma = n_a n_b + \frac{\beta}{\alpha} = n_a n_b - N \cdot \frac{n_a(n_b - n_{ab}) + n_b(n_a - n_{ab})}{(n_a - n_{ab}) + (n_b - n_{ab})}$$
$$= n_a n_b - N[n_a \rho + n_b(1-\rho)], \tag{22}$$

with $\rho = \frac{n_b - n_{ab}}{(n_a - n_{ab}) + (n_b - n_{ab})}$. Obviously, $0 \le \rho \le 1$.

Lemma 7 (Monotonicity, Case II) *If $n_{ab} < n_a = n_b$, then $f(t)$ is monotonically decreasing when $t < c_0$, and monotonically increasing when $t > c_0$, with c_0 given in (3).*

Proof If $n_a = n_b$, then $t^2 - 2c_0 t + \gamma = (t - c_0)^2$, and (21) becomes

$$f'(t) = \frac{-2\alpha}{v^3(t + n_a)v^3(t + n_b)} \cdot (t - c_0)^3.$$

so $f'(t) < 0$, if $t < c_0$; and $f'(t) > 0$ if $t > c_0$. □

Lemma 8 (Monotonicity, Case III) *If $n_a \ne n_b$, then $f(t)$ is monotonically increasing on $[c_0 - \sqrt{c_0^2 - \gamma}, c_0)$, and monotonically decreasing on $(c_0, c_0 + \sqrt{c_0^2 - \gamma}]$, with c_0 given in (3) and γ given in (22).*

Proof If $n_a < n_b$, then $\gamma \le n_a n_b - N n_a = -(N - n_b)n_a$, and

$$c_0^2 - \gamma \ge \left(\frac{N - n_a - n_b}{2}\right)^2 + (N - n_b)n_a$$
$$= [N^2 - 2N(n_a + n_b) + (n_a + n_b)^2 + 4N n_a - 4n_a n_b]/4$$
$$= [N^2 - 2N(n_b - n_a) + (n_b - n_a)^2]/4 = \left(\frac{N + n_a - n_b}{2}\right)^2 \ge 0. \tag{23}$$

Otherwise, when $n_a > n_b$, we have $\gamma \le n_a n_b - N n_b = -(N - n_a)n_b$, and

$$c_0^2 - \gamma \ge \left(\frac{N - n_a - n_b}{2}\right)^2 + (N - n_a)n_b$$
$$= [N^2 - 2N(n_a + n_b) + (n_a + n_b)^2 + 4N n_b - 4n_a n_b]/4$$
$$= [N^2 - 2N(n_a - n_b) + (n_a - n_b)^2]/4 = \left(\frac{N - n_a + n_b}{2}\right)^2 \ge 0. \tag{24}$$

From (23) and (24) we can see that in either case, $c_0^2 - \gamma \ge 0$. So there are two solutions to $t^2 - 2c_0 t + \gamma = 0$: $t_{1,2} = c_0 \pm \sqrt{c_0^2 - \gamma}$, and (21) reduces to

$$f'(t) = \frac{-2\alpha}{v^3(t + n_a)v^3(t + n_b)} \cdot (t - c_0)(t - t_1)(t - t_2).$$

We can prove that $[0, \Delta] \subseteq [t_1, t_2]$ as follows. If $n_a < n_b$, from (23) we know that $\sqrt{c_0^2 - \gamma} \ge \frac{N + n_a - n_b}{2}$. So

$$t_1 = c_0 - \sqrt{c_0^2 - \gamma} \le \frac{N - n_a - n_b}{2} - \frac{N + n_a - n_b}{2} = -n_a \le 0,$$

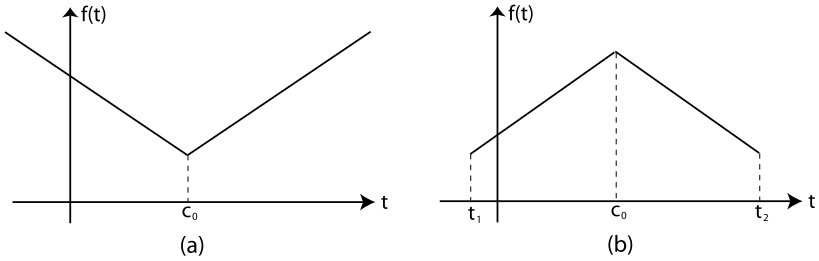


Fig. 4 The sketches of $f(t)$ when $\alpha < 0$. (**a**) $n_{ab} < n_a = n_b$ (Case II as discussed in Lemma 7); (**b**) $n_a \neq n_b$ (Case III as discussed in Lemma 8)

$$t_2 = c_0 + \sqrt{c_0^2 - \gamma} \geq \frac{N - n_a - n_b}{2} + \frac{N + n_a - n_b}{2} = N - n_b \geq \Delta.$$

On the other hand, if $n_a > n_b$, from (24) we know that $\sqrt{c_0^2 - \gamma} \geq \frac{N - n_a + n_b}{2}$. So

$$t_1 = c_0 - \sqrt{c_0^2 - \gamma} \leq \frac{N - n_a - n_b}{2} - \frac{N - n_a + n_b}{2} = -n_b \leq 0,$$
$$t_2 = c_0 + \sqrt{c_0^2 - \gamma} \geq \frac{N - n_a - n_b}{2} + \frac{N - n_a + n_b}{2} = N - n_a \geq \Delta.$$

Since $[0, \Delta] \subseteq [t_1, t_2]$, for any $t \in [0, \Delta]$, we have $t - t_1 \geq 0$ and $t - t_2 \leq 0$. Also note that $\alpha < 0$ and $t_1 < c_0 < t_2$. So $f'(t) > 0$, if $t < c_0$; and $f'(t) < 0$ if $t > c_0$. □

The monotonicity of $f(t)$ in Cases II and III can be illustrated in Fig. 4. Note that only trend lines are plotted, and the actual curves of $f(t)$ may not necessarily be straight line segments.

5.5 The tight evolving upper bound

In this subsection, we come up with the tight upper bound of ϕ_{ab}, based on the findings in the previous subsection. First, we briefly prove the symmetry of $f(t)$, which is helpful for us to compare the values of $f(0)$ and $f(\Delta)$, without actually computing them. Then we derive the upper bound corresponding to each case.

Lemma 9 (Symmetry) *$f(t)$ is symmetric about $t = c_0$.*

Proof A function $f(t)$ is symmetric about $t = c_0$ if $f(c_0 - x) = f(c_0 + x)$, forall x in its well-defined range. First, it is easy to see from (18) that $s(t)$ is symmetric about $t = c_0$. Then we can rewrite $f(t)$ in (16) into

$$f(t) = \frac{N(n_{ab} + t) - (n_a + t)(n_b + t)}{\sqrt{(t + n_a)(N - t - n_a)}\sqrt{(t + n_b)(N - t - n_b)}}$$
$$= \frac{Nn_{ab} - s(t)}{\sqrt{s(t) - Nn_a}\sqrt{s(t) - Nn_b}}.$$

So $f(t)$ is also symmetric about $t = c_0$. □

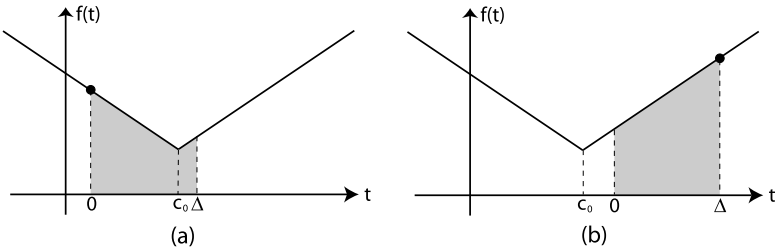


Fig. 5 Maximum value of $f(t)$ on $[0, \Delta]$ when $n_{ab} < n_a = n_b$

Lemma 10 (Upper Bound, Case I) *If* $n_{ab} = n_a = n_b$, *then* $\text{upper}(\phi_{ab}) = 1$.

Proof When $n_{ab} = n_a = n_b$, we have

$$f(t) = \frac{N(n_{ab}+t) - (n_a+t)(n_b+t)}{\sqrt{(n_a+t)(N-n_a-t)}\sqrt{(n_b+t)(N-n_b-t)}}$$
$$= \frac{N(n_a+t) - (n_a+t)^2}{(n_a+t)(N-n_a-t)} = 1, \forall t \in \mathcal{T}.$$

□

Lemma 11 (Upper Bound, Case II) *If* $n_{ab} < n_a = n_b$, *then*

$$\text{upper}(\phi_{ab}) = \begin{cases} f(0), & \text{if } n_a + n_b \le n; \\ f(\Delta), & \text{otherwise.} \end{cases}$$

Proof By analyzing $f(t)$ as in Lemma 7, the maximum value of $f(t)$ on $[0, \Delta]$ is either $f(0)$ or $f(\Delta)$. Figure 5(a) presents the case $c_0 \ge \frac{\Delta}{2}$. In this case, since 0 is not as close to c_0 than Δ, we have $f(0) \ge f(\Delta)$. Here $c_0 \ge \frac{\Delta}{2} \Leftrightarrow N - n_a - n_b \ge \Delta \Leftrightarrow n - n_a - n_b \ge 0 \Leftrightarrow n_a + n_b \le n$. Figure 5(b) presents the case $c_0 > \frac{\Delta}{2}$. In this case, since Δ is further away from c_0 than 0, we have $f(0) \le f(\Delta)$. □

Lemma 12 (Upper Bound, Case III) *If* $n_a \ne n_b$, *then*

$$\text{upper}(\phi_{ab}) = \begin{cases} f(\Delta), & \text{if } c_0 > \Delta; \\ f(0), & \text{if } c_0 < 0; \\ f(\lfloor c_0 \rfloor), & \text{otherwise.} \end{cases}$$

Proof By analyzing $f(t)$ as in Lemma 8, the maximum value of $f(t)$ on $[0, \Delta]$ is the largest among $f(0)$, $f(\Delta)$ and $f(c_0)$. Figure 6(a) presents the case $c_0 > \Delta$. In this case, since Δ is closer to c_0 than 0, we have $f(0) < f(\Delta)$. Figure 6(b) presents the case $0 \le c_0 \le \Delta$. In this case, obviously, $f(c_0)$ is the largest. However, $c_0 = \frac{N-n_a-n_b}{2}$ may not be an integer, so we take $\lfloor c_0 \rfloor$, which is the nearest integer to c_0. If $N - n_a - n_b$ is even, then $\lfloor c_0 \rfloor = c_0$; otherwise, it is equivalent to use $\lfloor c_0 \rfloor$ or $\lceil c_0 \rceil$, since they are equally apart from c_0 (see Lemma 9). Figure 6(c) presents the case $c_0 < 0$. In this case, since Δ is further away from c_0 than 0, we have $f(0) > f(\Delta)$. □

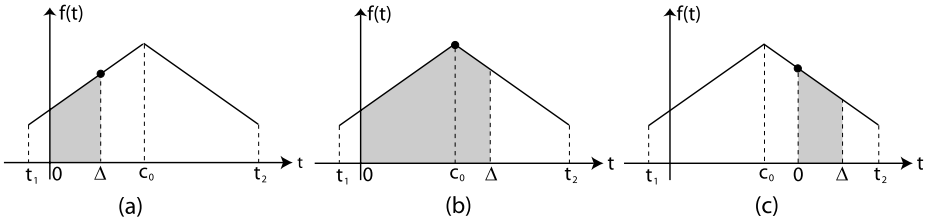


Fig. 6 Maximum value of $f(t)$ on $[0, \Delta]$ when $n_a \neq n_b$

In a nutshell, in this subsection we have derived the tight upper bound for ϕ_{ab} in the three different cases of pair $\{a, b\}$. In each case, we find a handful of candidate optimum points, at which upper(ϕ_{ab}) may reach its maximum value. By comparing the values of upper(ϕ_{ab}) at these points only, and choosing the maximum, we can eventually come up with the actual upper bound.

6 The CHECK-POINT+ algorithm

In our preliminary work (Zhou and Xiong 2008), we introduced a basic version of the volatile correlation computing algorithm, named CHECK-POINT, which was based on a loose upper bound of ϕ. In this paper, we design a new algorithm named CHECK-POINT+, which is an enhanced version of the CHECK-POINT algorithm. In the following, we describe the algorithm in detail.

The basic framework of CHECKPOINT+ is described in Algorithm 1. Given a minimum correlation threshold θ, and new transactions that are being collected, our goal is to run the ASP query after every S new transactions. For achieving this goal effectively, a checkpoint is set after every Δ new transactions. Typically, we have $S \ll \Delta$, meaning that ASP queries are much more frequent than checkpoints.

As time goes, we maintain CL, the list of candidate pairs, and I, the list of transactions in which each item exists. More specially, for each candidate pair in CL, we keep record of the item IDs (e.g., $\{a, b\}$) and its frequency in all transactions collected so far (e.g., n_{ab}). And for each item $a \in I$, we keep record of its item ID (i.e., a), its accumulative frequency (i.e., n_a), followed by the IDs of previous transactions that contain this item. Specially, we use I_a to represent the list of transactions which contain item a.

Algorithm 1 describes what happens when the k-th transaction has just been collected ($k = 1, 2, \ldots$). There are up to three steps: initializations, ASP query, and candidate list update. First, initializations are done by updating I and CL with the information contained in the new transaction T_k (Line 1). This includes appending the transaction ID, k, to each item in I that has been involved, and incrementing its pair-wise frequency, if any pair in CL co-occurs in T_k. Then, whenever S new transactions are collected since the last run of ASP query, we need to re-run the query to find the strong pairs up-to-date (Lines 2–7). For this purpose, all we need to do is to check the correlation of each pair on the candidate list (Line 3). All other pairs are safely ignored due to the way the candidate list was constructed (please refer to the checkpoint principle in Sect. 4). Since the frequency of each candidate pair is recorded, it is easy to compute the exact correlation directly (Line 4). Given the minimum correlation threshold θ, we output the pair as a strong one only if its correlation is no less than θ (Line 5). Note that candidate pairs is a superset of strong pairs, and a candidate pair may not always be strong for all ASP queries between two neighboring checkpoints.

Input: θ, minimal correlation threshold; T_k, the k-th transaction; CL, candidate list; I, item list; S, number of transactions between ASP queries; Δ, number of transactions between checkpoints.
Output: All strong pairs, whenever a ASP query is requested

```
1  Initializations with T_k;
2  if k mod S = 0 then
3  |  foreach item pair {a, b} ∈ CL do
4  |  |  φ_ab ← (nn_ab − n_a n_b) / sqrt(n_a(n − n_a) n_b(n − n_b));
5  |  |  if φ_ab ≥ θ then output {a, b} as a strong pair
6  |  end
7  end
8  if k mod Δ = 0 then
9  |  CL ← UpdateCandidateList(CL, I, θ, Δ)
10 end
```

Algorithm 1: The CHECK-POINT+ Algorithm

Finally, for every Δ transactions, we update the candidate list (Lines 8–10) by calling a sub-procedure named UpdateCandidateList (Line 9).

Procedure 2 illustrates what happens at each checkpoint. Given the up-to-date candidate list CL, item list I, and the necessary parameters, the goal is to rebuild the list of candidates, CL', by taking into account of future transactions up to the next checkpoint. In this procedure, we need to screen each possible pair of items to see if it remains or has become a candidate. More specifically speaking, for any pair of items $\{a, b\}$ (Line 2), if $n_a = n_b$, then it may correspond to Cases I or II (Lines 4–6); otherwise it corresponds to Case III (Lines 8–15). After determining the upper bound by case, as stated in Lemmas 10 through 12, we append the pair to the new candidate list CL' if its upper bound is no less than θ (Line 17).

In the case that $n_a = n_b$ (Line 3), we first find n_{ab} by either looking it up from the old candidate list (if available), or counting it from scratch by comparing I_a and I_b (Line 4). Further, if $n_{ab} = n_a$, then this pair corresponds to Case I (Lemma 6), and the upper bound is 1, according to Lemma 10. Since a valid choice of θ has to be in [0, 1], this upper bound is definitely no less than θ, and so the pair is appended as a candidate immediately and we can proceed to the next pair (Line 5). Otherwise, the pair must correspond to Case II (Lemma 7). In this case, we can calculate the upper bound according to Lemma 11 (Line 6).

In the other case, we have $n_a \neq n_b$ (Line 7), and it corresponds to Case III (Lemma 8). In this case, we first try to look up n_{ab} from the old candidate list CL (Line 8). If $\{a, b\}$ is not in CL, then we may need to count the exact n_{ab} by comparing I_a versus I_b directly (Line 12). Since this is time consuming, we employ the upper bound to do some preliminary pruning (Lines 10–11). The idea is to fit in n_{ab} with its maximal possible value, i.e. $\min\{n_a, n_b\}$, and calculate the upper bound as we derived in Lemma 12 (Line 10). Since we are using a possibly larger value than the actual n_{ab}, this upper bound is no less than the actual upper bound. If this larger bound is lower than the threshold, then the pair can be safely pruned (Line 11). Otherwise, go on to count the exact n_{ab} (Line 12) to be used for possible refinement. If we find that n_{ab} really equals $\min\{n_a, n_b\}$, as we assumed in Line 10, then the previously calculated maximum value is the exact bound value, and we know it is a candidate since it is not pruned in Line 11. So we append this pair as a candidate and continue with the next pair (Line 13). When we reach Line 15, it can be the initial calculation for a pair that is

```
1  CL' ← ∅;
2  foreach pair of different items a, b ∈ I do
3      if n_a = n_b then
4          if {a, b} ∈ CL then look up n_ab from CL; else n_ab ← #(I_a ∩ I_b);
5          if n_ab = n_a then append {a, b} to CL' and continue;
6          max ← upper bound by Lemma 11;
7      else
8          if {a, b} ∈ CL then look up n_ab from CL;
9          else
10             n_ab ← min{n_a, n_b} and max ← upper bound by Lemma 12;
11             if max < θ then continue;
12             n_ab ← #(I_a ∩ I_b);
13             if n_ab = min{n_a, n_b} then append {a, b} to CL' and continue;
14         end
15         max ← upper bound by Lemma 12;
16     end
17     if max ≥ θ then append {a, b} to CL';
18 end
19 return CL'
```

Procedure `UpdateCandidateList(`CL, I, θ, Δ`)`

previously a candidate (n_{ab} found in Line 8), or the refinement of the upper bound for a pair that is not pruned in Line 11 (and the exact n_{ab} is found in Line 12). Now we can calculate the upper bound with the exact n_{ab} by following Lemma 12 (Line 15).

Finally, once we reach Line 17, the calculated upper bound is compared with the correlation threshold θ. Item pair $\{a, b\}$ is output into CL' as a candidate if its upper bound is no less than θ.

The major difference between CHECK-POINT+ and CHECK-POINT is that we use a tight evolving upper bound in CHECK-POINT+ for better pruning effect. Also, we try to avoid counting the exact pair-wise frequency by finding them directly from the previous candidate list (to re-use more intermediate results) and using its maximal possible value (to provide additional pruning).

7 Experimental results

In this section, we first briefly introduce three benchmark algorithms, against which we compare the CHECK-POINT+ algorithm. Then, we present the experimental setup. Finally, we show the evaluation results with respect to computation efficiency, space usage, and sensitivity to various parameters.

7.1 Benchmark algorithms

In order to evaluate the performance of the CHECK-POINT+ algorithm, we compare it with several benchmark algorithms. Specifically, there are three relevant algorithms, r-TAPER, SAVE-ALL, and CHECK-POINT, as described in Zhou and Xiong (2008). In the following, we briefly summarize these algorithms.

r-TAPER A straightforward way to compute volatile correlation is to re-compute correlations for each ASP query request. Along this line, a quick solution is to employ an efficient static correlation computing algorithm, for instance, TAPER (Xiong et al. 2004), for each round of correlation computing. We call this method r-TAPER, meaning "repeated TAPER."

SAVE-ALL A computational bottleneck for ASP correlation computing is to count the frequency of each item pair on the fly. This fact has motivated the SAVE-ALL algorithm, which trades space for time. SAVE-ALL stores the frequencies of all item pairs and incrementally updates the stored values while new data are added into the database. In this way, SAVE-ALL uses extra space for the sake of saving computation time. The limitation of this method is that when the number of items becomes very large, we may not have enough memory space for running this algorithm.

CHECK-POINT The above mentioned two algorithms represent two extreme cases of the ASP correlation query problem in dynamic data environments. The r-TAPER algorithm, which repeat the query every time when new data become available, disregards the previously computed results, and thus leads to high-computational cost. On the other hand, the SAVE-ALL algorithm requires an extremely large amount of memory space for saving the intermediate computing results. This becomes infeasible when the number of items is very large. The CHECK-POINT algorithm, however, seeks a solution in between. As provided in our previous work (Zhou and Xiong 2008), the CHECK-POINT algorithm relies on a loose upper bound of evolving correlation values for selecting candidate item pairs.

7.2 The experimental setup

The CHECK-POINT algorithm, along with the above mentioned benchmark algorithms, have been implemented for experimental evaluations. In this subsection, we describe the experimental setup, including datasets, parameters, and the platform.

7.2.1 Datasets

We use datasets from the Frequent Itemset Mining Implementations (FIMI) Repository Web site (http://fimi.cs.helsinki.fi/data/). These datasets are often used as benchmarks for evaluating frequent pattern mining algorithms. We have deliberatively chosen datasets with various sizes and densities, and Table 3 summarizes the basic statistics of the datasets we have used.

Since our goal is to perform the ASP correlation queries in dynamic data environments, we split each dataset into two parts. The first part consists of, approximately, the first two thirds of all transactions in the original dataset, and we call it the "base dataset". The second part consists of the remaining transactions, and we call it the "incremental dataset". Each run of the experiments start with the base dataset, load new transactions from the incremental dataset one by one, and do the queries and updates whenever applicable. Despite splitting the data, we load transactions in their original order stored in their respective data files. This

Table 3 Experimental datasets

Dataset	# Items	# Pairs	# Transactions
Chess	75	2,775	3,196
Mushroom	119	7,021	8,124
Connect	129	8,256	67,557
Pumsb	2,113	2,231,328	49,046

Table 4 Experimental parameters

Name	Symbol	Definition	Values
Correlation threshold	θ	The minimum value above which an item pair's correlation is called "strong".	0.3, **0.5**, 0.7, 0.9
Step size	S	The number of transactions between ASP queries.	1, 2, **5**, 10
Incremental ratio	α	The number of transactions between checkpoints (Δ), divided by the number of transactions accumulated (n).	0.001, **0.01**, 0.05, 0.1

process is mimicking the real-world applications where new transactions are continually collected and processed in a large database.

7.2.2 Parameters

During the experiments, we have tried various combinations of parameters. Table 4 lists the experimental parameters we have considered.

The correlation threshold θ and the step size S are common parameters across all four algorithms. We try $\theta = 0.3$, $\theta = 0.5$, $\theta = 0.7$, and $\theta = 0.9$, since positive correlations are typical for ASP queries. Considering real world applications, the smaller step sizes the better, which means that the ASP queries are updated in a timely manner. Ideally, the step size is 1, meaning that whenever a new transaction is collected, the ASP query is updated right away. As a result, in the experiments, we try small step sizes, such as $S = 1$, $S = 2$, $S = 5$, $S = 10$. The incremental ratio $\alpha = \Delta/n$ is a measure related to checkpoint density, since the larger α is, the more sparse are the checkpoints. Obviously, the parameter α only applies to CHECK-POINT and CHECKPOINT+, for which we need to determine where to set the checkpoints. For the experiments, we try $\alpha = 0.001$, $\alpha = 0.01$, $\alpha = 0.05$, and $\alpha = 0.1$. The bold values in Table 4 are the comparison baselines by default. In the experiments, we choose $\theta = 0.5$, $S = 5$, $\alpha = 0.01$ unless otherwise specified.

7.2.3 Platform

All the experiments are performed on a Dell Optiplex 755 Minitower, with Intel 2 Quad processor Q6600 and 4 GB of memory, and running the Microsoft Windows XP Professional operation system.

7.3 Computation efficiency

In this subsection, we focus on comparing the computation efficiency of CHECK-POINT+ against benchmark algorithms, and study the sensitivity of the performance with respect to different parameters.

Figure 7 shows the accumulative running time of different algorithms on each dataset. In each sub-figure, the x-axis is transaction ID, and the y-axis is the accumulative running time of the ASP queries, which has been taken natural log to shrink the deviation of larger values. We find that in general, r-TAPER is the slowest, SAVE-ALL is the fastest, while CHECK-POINT and CHECK-POINT+ are in between. However, CHECK-POINT+ runs more efficiently than CHECK-POINT.

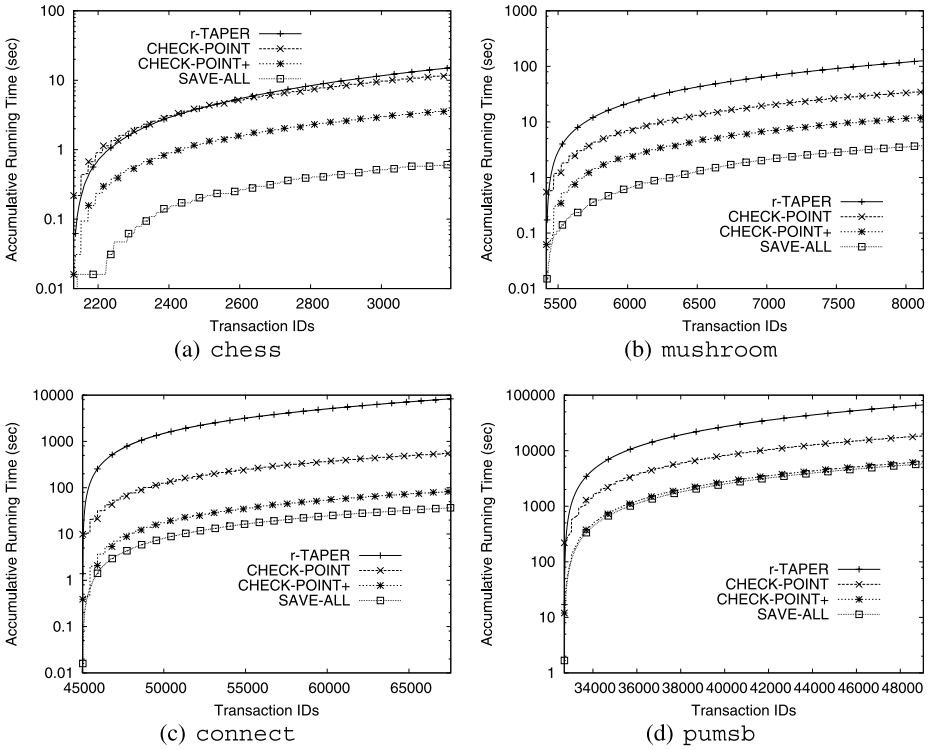


Fig. 7 Accumulative running time

The results shown in Fig. 7 are based on default parameter values ($\theta = 0.5$, $S = 5$, $\alpha = 0.01$). In the following, we study the sensitivity of the computation efficiency with respect to different parameters. Figures 8 through 10 demonstrate comparisons of the average execution time of a query by different algorithms. Here, the average query time of a query is defined as the accumulative running time at the end of the experiment, divided by the number of ASP queries executed.

Figure 8 shows the sensitivity of computation efficiency for different correlation thresholds. In each sub-figure, there are four groups of vertical bars, corresponding to various correlation thresholds ($\theta = 0.3$, $\theta = 0.5$, $\theta = 0.7$, and $\theta = 0.9$). From Fig. 8, we have the following observations.

- When the threshold decreases, the average query time of r-TAPER and CHECK-POINT+ increases, but the change of CHECK-POINT+ is much less dramatic. In other words, CHECK-POINT+ is much less sensitive to the correlation threshold θ than r-TAPER.
- The average query time of CHECK-POINT+ is much smaller than that of CHECK-POINT for each dataset at different thresholds. The average query time for CHECK-POINT does not change significantly for some thresholds, since the upper bound used in CHECK-POINT is loose and has less pruning effect, resulting in more work in building and maintaining the candidate list.
- SAVE-ALL is almost always the fastest. However, there are minor issues related to implementation, as discussed below. First, despite the extra work for building candidate lists, CHECK-POINT+ sometimes performs better than SAVE-ALL on the dataset pumsb

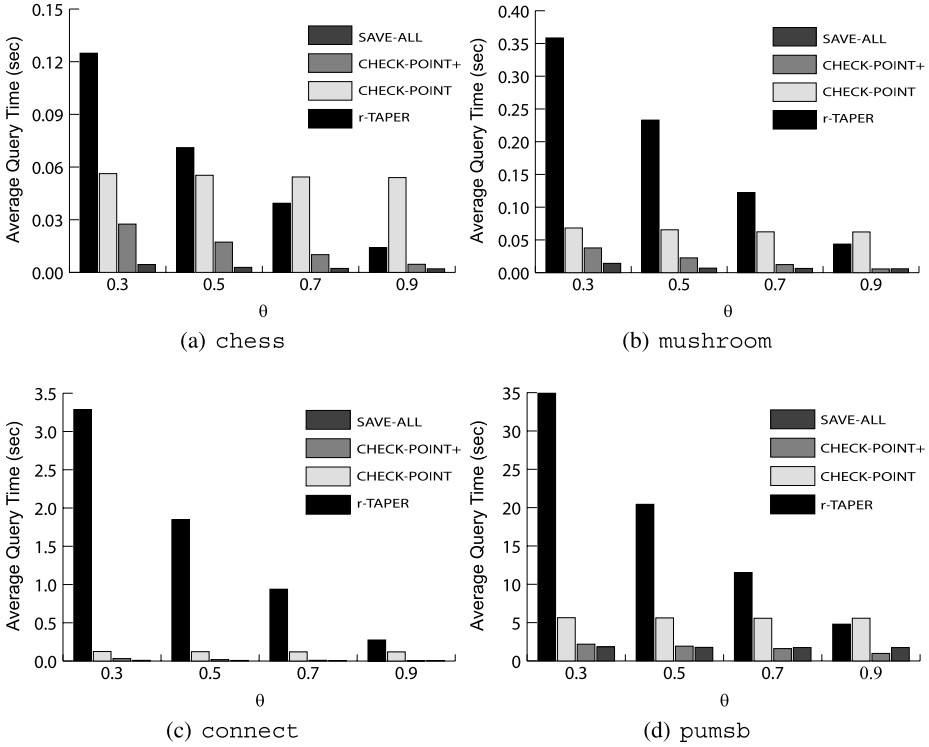


Fig. 8 Average query time with different correlation thresholds

when using large thresholds ($\theta = 0.7$ and $\theta = 0.9$). The reason is that with higher correlation thresholds, it is relatively faster to build the candidate list, and the list tends to be very small, compared to the number of all pairs. As a result, it becomes easier to build and maintain the candidate list and search for strong pairs within the small number of candidates (CHECK-POINT+) instead of computing to determine strong ones from all pairs (SAVE-ALL). As we can see, such scenario is especially applicable to datasets with a large number of items and high correlation thresholds. Second, although SAVE-ALL is expected to take constant time for different thresholds on each dataset, sometimes we observe longer time for lower thresholds due to substantially increased size of ASP query output. For example, when we run SAVE-ALL on `mushroom` using $\theta = 0.3$, about 10% of all pairs are strong and need to be written into the output file. However, when using $\theta = 0.5$, this number reduces quickly to 5%. Therefore, the time for SAVE-ALL in our experiments may be affected by the time taken for outputs.

Figure 9 shows the sensitivity of computation efficiency for different step sizes. In each sub-figure, there are four groups of vertical bars, corresponding to various step sizes ($S = 1$, $S = 2$, $S = 5$, and $S = 10$). From Fig. 9, we have the following observations.

– When the step size increases, the average query time of CHECK-POINT and CHECK-POINT+d increases accordingly. However, the change of CHECK-POINT+ is much less dramatic than CHECK-POINT. In other words, CHECK-POINT+ is less sensitive than CHECK-POINT to different step sizes.

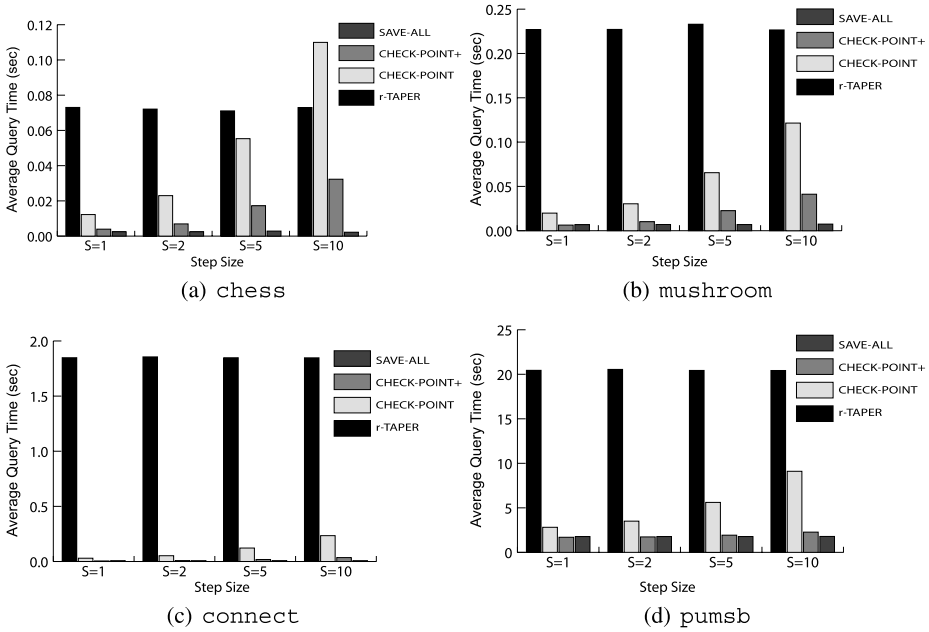


Fig. 9 Average query time with different step sizes

– r-TAPER takes longest time and SAVE-ALL remains the fastest. This is self-explanatory, since r-TAPER does not re-use intermediate results, while SAVE-ALL re-uses the frequency of all pairs.
– For CHECK-POINT and CHECK-POINT+, generally, larger step sizes correspond to longer query time on average. The reason is that with larger step sizes, there are fewer queries executed between two checkpoints. According to our findings in Zhou and Xiong (2008), building the candidate list is the main source of computation cost for checkpoint-based algorithms. In other words, the time for building candidate lists are amortized among fewer queries, resulting in longer query time on average.

From the above comparisons, since SAVE-ALL is simple and fast, one may argue that SAVE-ALL is still a better choice than CHECK-POINT+ in some settings (e.g., infinite data streams). This is true, especially for datasets with a small number of items (so that the number of all pairs will be small). However, in many applications, the number of all pairs is so large that SAVE-ALL becomes impractical. Such situation is very common in today's data intensive applications. Moreover, as observed with the `pumsb` dataset, sometimes CHECK-POINT+ may perform better than or equivalently to SAVE-ALL in practice.

Figure 10 shows the sensitivity of computation efficiency for various checkpoint densities. In each sub-figure, there are four groups of vertical bars, corresponding to different checkpoint densities ($\alpha = 0.001$, $\alpha = 0.01$, $\alpha = 0.05$, and $\alpha = 0.1$). Since this parameter only applies to CHECK-POINT and CHECK-POINT+, in each parameter group there are two vertical bars instead of four. We can see that when the checkpoint density increases (the value of α decreases), the average query time of CHECK-POINT and CHECK-POINT+ increases, but the change of CHECK-POINT+ is much less dramatic. In other words, CHECK-POINT+ is less sensitive than CHECK-POINT to the change of checkpoint density.

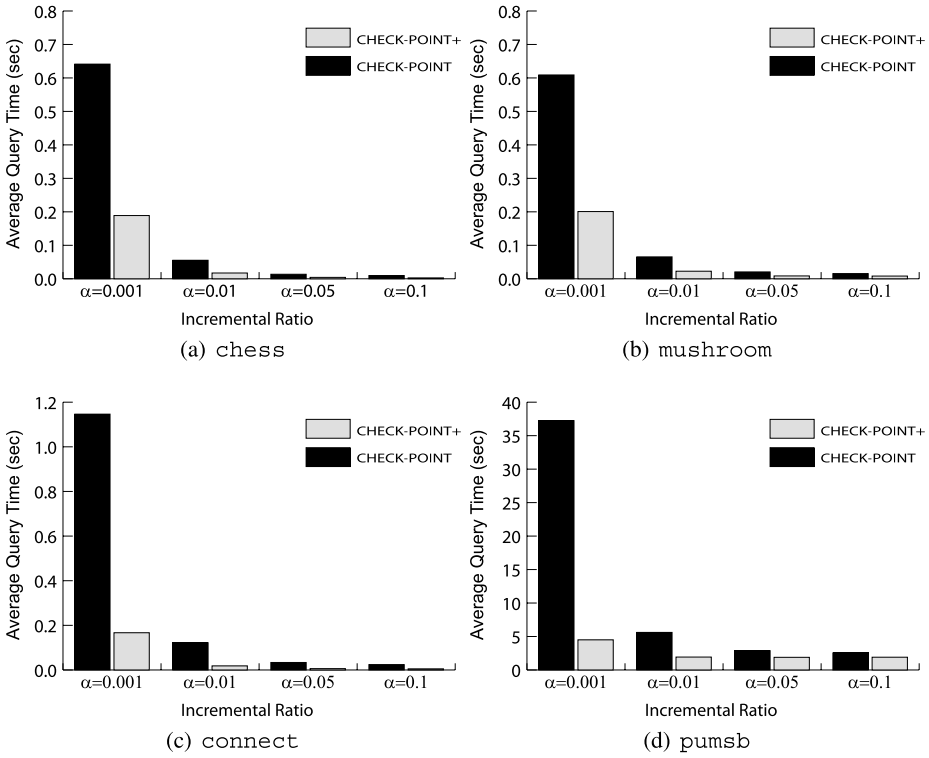


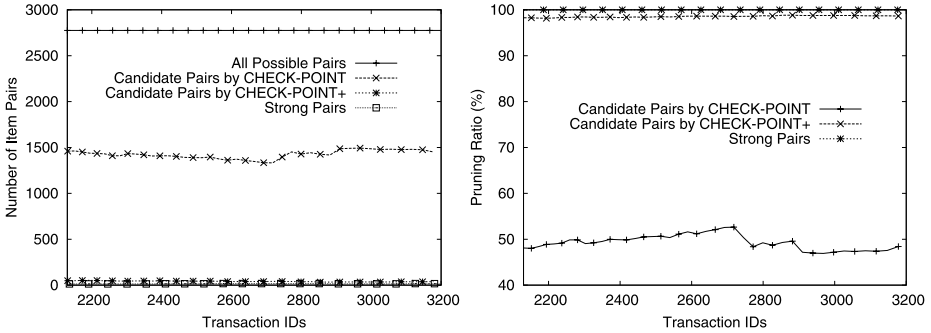
Fig. 10 Average query time with different incremental ratios

7.4 Use of space

In this subsection, we investigate the space requirement of the CHECK-POINT+ algorithm. Along this line, our goal is to check how many item pairs are selected as candidates, so that their frequencies need to be stored and maintained. Specifically, we focus on the number of candidate pairs by CHECK-POINT and CHECK-POINT+ only, because the space requirements for r-TAPER and SAVE-ALL are constant for a given dataset. Also, we include the numbers of all possible pairs and strong pairs as baselines.

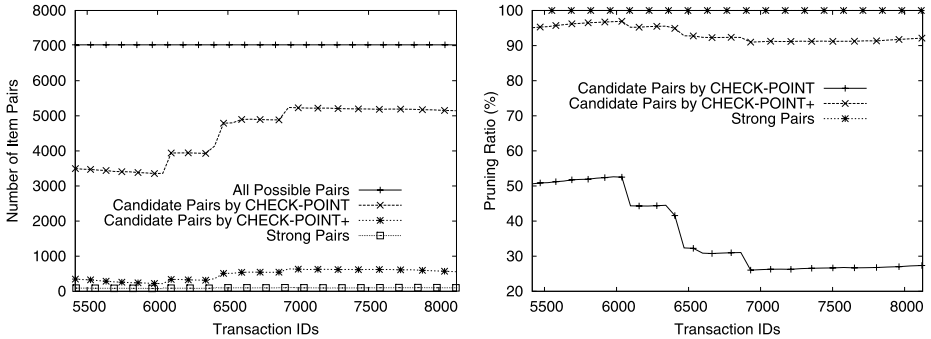
Figures 11, 12, 13 and 14 illustrate comparisons on the use of space for each dataset. In each of them, there are two sub-figures. The one on the left shows the numbers of pairs, and the one on the right shows the corresponding pruning ratios (PR). The pruning ratio (PR) is the proportion of pairs that are pruned, and is calculated as one minus the ratio of the number of candidate pairs (or strong pairs) to the number of all possible pairs. From Figs. 11 through 14 we have the following observations.

- CHECK-POINT+ has significantly better pruning effect than CHECK-POINT; that is, CHECK-POINT+ requires much less space than CHECK-POINT.
- CHECK-POINT+ has a extremely high pruning ratio (more than 90%), except for `pumsb`. This indicates that the performance of CHECK-POINT+ may be data dependent, and we may need to select different parameters for different datasets. The default



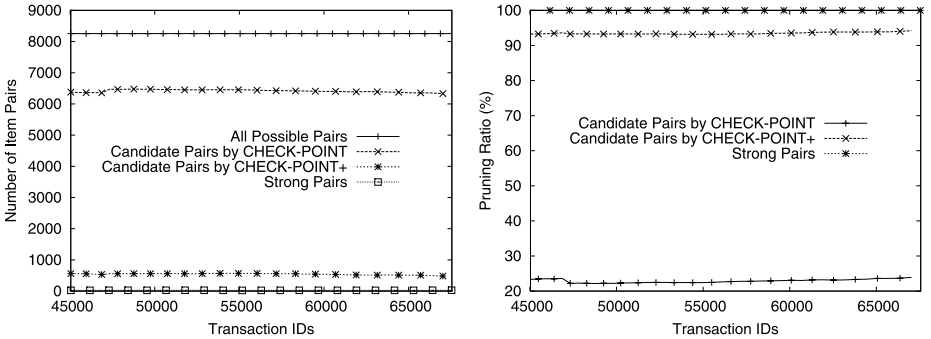
(a) Candidate List Size (b) Pruning Ratio

Fig. 11 Use of space by the CHECK-POINT and CHECK-POINT+ on `chess`



(a) Candidate List Size (b) Pruning Ratio

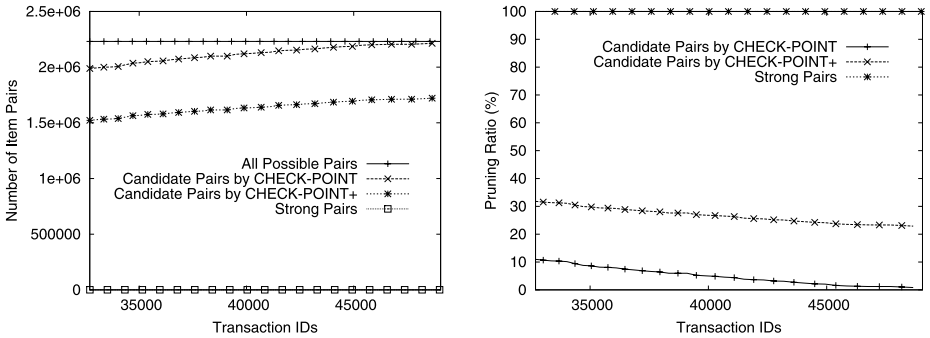
Fig. 12 Use of space by the CHECK-POINT and CHECK-POINT+ on `mushroom`



(a) Candidate List Size (b) Pruning Ratio

Fig. 13 Use of space by the CHECK-POINT and CHECK-POINT+ on `connect`

parameter ($\alpha = 0.01$) may be good enough for `chess`, `mushroom`, and `connect`, but not so good for `pumsb`. By examining the data characteristics more closely, we find that the rank-support distribution of `pumsb` is extremely skewed.



(a) Candidate List Size (b) Pruning Ratio

Fig. 14 Use of space by the CHECK-POINT and CHECK-POINT+ on `pumsb`

With an extremely large number of low-support items, CHECK-POINT+ may become less efficient. The reason is that given the same computation settings (same n and Δ), the upper bound of the correlation between a pair of low-support items may be more loose than that of a high-support one, since a small number of new transactions are able to affect their correlation values greatly in extreme cases. For example, suppose in the first 1000 transactions, $n_a = n_b = 1$ and $n_{ab} = 0$. Then currently $\phi_{ab} \approx -0.01$. Suppose $\Delta = 10$. In an extreme case, the 10 new transactions all contain both items a and b. Then the correlation will become $\approx$0.908. In other words, since the upper bound becomes so high, that such low-support pairs are hard to prune. To tackle this problem, we can take two approaches in practice. The first approach is to simply ignore low-support items. This method may not guarantee completeness all the time, but if sharp changes do occur, the missing items may still be picked up at the next checkpoint. Removal of low support items is very common in frequent pattern mining (Agrawal et al. 1993). The second approach is to treat low-support items separately from other items. For instance, we make a special list of low-support items. Since they happened less frequently previously, each of them has a very short list of transaction IDs. This is similar to the idea of mixed data structure for frequent itemset counting (Uno et al. 2005).

8 Conclusions

In this paper, we studied the problem of correlation computing in large and dynamically growing data sets. Specifically, we designed a CHECK-POINT+ algorithm for dynamically searching all the item pairs with correlations above a user-specified threshold. The key idea is to establish a computation buffer by setting a checkpoint for dynamic input data. This checkpoint can be exploited to identify a list of candidate pairs, which are maintained and whose correlations are computed, as new transactions are added into the database. However, if the total number of new transactions is beyond the checkpoint, a new candidate list is generated by the new checkpoint. All the checkpoints have been established by a tight evolving correlation upper bound, which shows a local monotonicity property.

Moreover, we carried out a number of experiments to evaluate the performance of the CHECK-POINT+ algorithm. Experimental results on real-world data sets show that CHECK-POINT+ can compact the use of memory space by maintaining a reduced candidate pair list, which is only a very small portion of all the item pairs. Also, CHECK-POINT+

can significantly reduce the correlation computing cost in dynamic data sets with a large number of transactions and has a much better computational performance than state-of-the-art benchmark methods. Finally, the experimental results also show that CHECK-POINT+ is less sensitive to the change of parameters compared to benchmark algorithms.

Acknowledgements This research was partially supported by the National Science Foundation (NSF) via grant number CNS 0831186, the Rutgers CCC Green Computing Initiative, and the National Natural Science Foundation of China (70890080).

Appendix A: Deriving $\frac{\partial f(x,y;c)}{\partial y}$ in (11)

In the following and subsequent proofs, note that given $v(x)$ in (2), we have $v'(x) = \frac{N/2-x}{v(x)}$. From (9) we have $f(x, y; c) = \frac{x(N-y)-c}{v(x)v(y)} = \frac{1}{v(x)} \cdot \frac{(Nx-c)-xy}{v(y)}$, so

$$\begin{aligned}
\frac{\partial f(x, y; c)}{\partial y} &= \frac{1}{v(x)} \cdot \frac{(-x) \cdot v(y) - [(Nx - c) - xy] \cdot v'(y)}{v^2(y)} \\
&= \frac{(-x) \cdot v(y) - [(Nx - c) - xy] \cdot \frac{N/2-y}{v(y)}}{v(x)v^2(y)} \\
&= \frac{(-x) \cdot v^2(y) - [(Nx - c) - xy] \cdot (N/2 - y)}{v(x)v^3(y)} \\
&= \frac{-2xy(N - y) - [x(N - y) - c](N - 2y)}{2v(x)v^3(y)} \\
&= \frac{-2xy(N - y) - x(N - y)(N - 2y) + c(N - 2y)}{2v(x)v^3(y)} \\
&= \frac{-Nx(N - y) + c(N - y - y)}{2v(x)v^3(y)} = \frac{(c - Nx)(N - y) - cy}{2v(x)v^3(y)}.
\end{aligned} \tag{25}$$

Appendix B: Deriving $\frac{\partial f(y,x;c)}{\partial y}$ in (12)

From (9) we have $f(y, x; c) = \frac{y(N-x)-c}{v(x)v(y)} = \frac{1}{v(x)} \cdot \frac{(N-x)y-c}{v(y)}$, so

$$\begin{aligned}
\frac{\partial f(y, x; c)}{\partial y} &= \frac{1}{v(x)} \cdot \frac{(N - x) \cdot v(y) - [(N - x)y - c] \cdot v'(y)}{v^2(y)} \\
&= \frac{(N - x) \cdot v(y) - [(N - x)y - c] \cdot \frac{N/2-y}{v(y)}}{v(x)v^2(y)} \\
&= \frac{(N - x) \cdot v^2(y) - [(N - x)y - c] \cdot (N/2 - y)}{v(x)v^3(y)} \\
&= \frac{2(N - x) \cdot y(N - y) - [(N - x)y - c] \cdot (N - 2y)}{2v(x)v^3(y)} \\
&= \frac{(N - x)y(2N - 2y) - (N - x)y(N - 2y) + c(N - 2y)}{2v(x)v^3(y)} \\
&= \frac{N(N - x)y + c(N - 2y)}{2v(x)v^3(y)} = \frac{[N(N - x) - c]y + c(N - y)}{2v(x)v^3(y)}.
\end{aligned} \tag{26}$$

Appendix C: Deriving $f'(t)$ in (17)

First, we have $w(t) = N(n_{ab} + t) - (n_a + t)(n_b + t) = -t^2 + 2c_0 t + c_1$, where c_0 is given in (3) and $c_1 = Nn_{ab} - n_a n_b$. So $w'(t) = 2(c_0 - t)$. Let $m(t) = v^{-1}(t + n_a)v^{-1}(t + n_b)$, then

$$\begin{aligned} m'(t) &= -\frac{v'(t+n_a)}{v^2(t+n_a)} \cdot \frac{1}{v(t+n_b)} - \frac{1}{v(t+n_a)} \cdot \frac{v'(t+n_b)}{v^2(t+n_b)} \\ &= -\frac{\frac{N}{2} - (t+n_a)}{v^3(t+n_a)} \cdot \frac{1}{v(t+n_b)} - \frac{1}{v(t+n_a)} \cdot \frac{\frac{N}{2} - (t+n_b)}{v^3(t+n_b)} \\ &= \frac{(t+n_a-\frac{N}{2})v^2(t+n_b) + (t+n_b-\frac{N}{2})v^2(t+n_a)}{v^3(t+n_a)v^3(t+n_b)}, \end{aligned} \tag{27}$$

where the numerator reduces to

$$\begin{aligned} &(t+n_a-\frac{N}{2})(t+n_b)(N-t-n_b) + \left(t+n_b-\frac{N}{2}\right)(t+n_a)(N-t-n_a) \\ &= (t+n_a)(t+n_b)(N-t-n_b) - \frac{N}{2}(t+n_b)(N-t-n_b) \\ &\quad + (t+n_b)(t+n_a)(N-t-n_a) - \frac{N}{2}(t+n_a)(N-t-n_a) \\ &= (t+n_a)(t+n_b)(2N-2t-n_a-n_b) - \frac{N}{2}\left[N(2t+n_a+n_b) - (t+n_b)^2 - (t+n_a)^2\right] \\ &= N(t+n_a)(t+n_b) + (t+n_a)(t+n_b)(N-2t-n_a-n_b) \\ &\quad - N^2\left(t+\frac{n_a+n_b}{2}\right) + \frac{N}{2}[(t+n_a)^2 + (t+n_b)^2] \\ &= (t+n_a)(t+n_b)(N-2t-n_a-n_b) - N^2\left(t+\frac{n_a+n_b}{2}\right) + 2N\left(t+\frac{n_a+n_b}{2}\right)^2 \\ &= (t+n_a)(t+n_b)(N-2t-n_a-n_b) + N\left(t+\frac{n_a+n_b}{2}\right)(2t+n_a+n_b-N) \\ &= (2t-N+n_a+n_b)\left[N\left(t+\frac{n_a+n_b}{2}\right) - (t+n_a)(t+n_b)\right] \\ &= 2\left(t-\frac{N-n_a-n_b}{2}\right)\cdot\left[-t^2 + (N-n_a-n_b)t + \frac{N(n_a+n_b)}{2} - n_a n_b\right] \\ &= 2(t-c_0)(-t^2+2c_0 t+c_2), \end{aligned} \tag{28}$$

with $c_2 = \frac{N(n_a+n_b)}{2} - n_a n_b$. Since $f(t) = w(t)m(t)$, the first derivative of $f(t)$ will be

$$\begin{aligned} f'(t) &= w'(t)m(t) + w(t)m'(t) \\ &= 2(c_0-t)\cdot\frac{1}{v(t+n_a)v(t+n_b)} + (-t^2+2c_0 t+c_1)\cdot\frac{2(t-c_0)(-t^2+2c_0 t+c_2)}{v^3(t+n_a)v^3(t+n_b)} \\ &= \frac{2(c_0-t)[v^2(t+n_a)v^2(t+n_b) - (-t^2+2c_0 t+c_1)(-t^2+2c_0 t+c_2)]}{v^3(t+n_a)v^3(t+n_b)} \end{aligned}$$

$$= \frac{2(c_0 - t)[k_1(t) - k_2(t)]}{v^3(t + n_a)v^3(t + n_b)}, \tag{29}$$

where

$$\begin{aligned} k_1(t) &= v^2(t + n_a)v^2(t + n_b) = (t + n_a)(N - t - n_a)(t + n_b)(N - t - n_b) \\ &= [(t + n_a)(t - N + n_b)] \cdot [(t - N + n_a)(t + n_b)] \\ &= (t^2 - 2c_0t + n_an_b - Nn_a)(t^2 - 2c_0t + n_an_b - Nn_b) \\ &= [s(t) - Nn_a][s(t) - Nn_b] = s^2(t) - N(n_a + n_b) \cdot s(t) + N^2n_an_b, \end{aligned} \tag{30}$$

$$\begin{aligned} k_2(t) &= (-t^2 + 2c_0t + c_1)(-t^2 + 2c_0t + c_2) = (t^2 - 2c_0t - c_1)(t^2 - 2c_0t - c_2) \\ &= [s(t) - Nn_{ab}][s(t) - N(n_a + n_b)/2] \\ &= s^2(t) - N\left(n_{ab} + \frac{n_a + n_b}{2}\right) \cdot s(t) + N^2n_{ab} \cdot \tfrac{n_a + n_b}{2}, \end{aligned} \tag{31}$$

with $s(t) = t^2 - 2c_0t + n_an_b$. From (30) and (31), it is easy to see that

$$k_1(t) - k_2(t) = N\left(n_{ab} - \frac{n_a + n_b}{2}\right) \cdot s(t) + N^2\left(n_an_b - n_{ab}\frac{n_a + n_b}{2}\right). \tag{32}$$

Substituting (32) into (29), we have the right hand side in (17).

References

Agrawal, R., Imielinski, T., & Swami, A. N. (1993). Mining association rules between sets of items in large databases. In P. Buneman & S. Jajodia (Eds.), *Proceedings of the 1993 ACM SIGMOD international conference on management of data* (pp. 207–216). Washington, D.C., USA, May 26–28, 1993. New York: Assoc. Comput. Math.

Alexander, C. (2001). *Market models: a guide to financial data analysis*. London: Wiley.

Bayardo, R. J., & Agrawal, R. (1999). Mining the most interesting rules. In U. Fayyad, S. Chaudhuri, & D. Madigan (Eds.), *Proceedings of the Fifth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 145–154). San Diego, California, USA, August 15–18, 1999. New York: Assoc. Comput. Math.

Bentley, J. (2000). *Programming pearls* (2nd ed.). Reading: Addison–Wesley.

Brin, S., Motwani, R., & Silverstein, C. (1997). Beyond market baskets: generalizing association rules to correlations. In J. Peckham (Ed.), *Proceedings of the 1997 ACM SIGMOD international conference on management of data* (pp. 265–276). May 13–15, 1997, Tucson, Arizona, USA. New York: Assoc. Comput. Math.

Cohen, P., Cohen, J., West, S. G., & Aiken, L. S. (2002). *Applied multiple regression/correlation analysis for the behavioral sciences* (3rd ed.). Hillstale: Erlbaum.

DuMouchel, W., & Pregibon, D. (2001). Empirical Bayes screening for multi-item associations. In F. Provost & R. Srikant (Eds.), *Proceedings of the Seventh ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 67–76). San Francisco, California, USA, August 26–29, 2001. New York: Assoc. Comput. Math.

Ilyas, I. F., Markl, V., Haas, P. J., Brown, P., & Aboulnaga, A. (2004). CORDS: Automatic discovery of correlations and soft functional dependencies. In G. Weikum, A. C. König, & S. Deßloch (Eds.), *Proceedings of the 2004 ACM SIGMOD international conference on management of data* (pp. 647–658). Paris, France, June 13–18, 2004. New York: Assoc. Comput. Math.

Jermaine, C. (2001). The computational complexity of high-dimensional correlation search. In N. Cercone, T. Y. Lin, & X. Wu (Eds.), *Proceedings of the 2001 IEEE international conference on data mining* (pp. 249–256). November 29–December 2, 2001, San Jose, California, USA. Los Alamitos: IEEE Computer Society.

Jermaine, C. (2003). Playing hide-and-seek with correlations. In L. Getoor, T. E. Senator, P. Domingos, & C. Faloutsos (Eds.), *Proceedings of the Ninth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 559–564). Washington, D.C., August 24–27, 2003. New York: Assoc. Comput. Math.

Ke, Y., Cheng, J., & Ng, W. (2006). Mining quantitative correlated patterns using an information-theoretic approach. In T. Eliassi-Rad, L. H. Ungar, M. Craven, & D. Gunopulos (Eds.), *Proceedings of the Twelfth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 227–236). Philadelphia, Pennsylvania, USA, August 20–23, 2006. New York: Assoc. Comput. Math.

Ke, Y., Cheng, J., & Ng, W. (2007). Correlation search in graph databases. In P. Berkhin, R. Caruana, & X. Wu (Eds.), *Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 390–399). San Jose, California, USA, August 12–15, 2007. New York: Assoc. Comput. Math.

Kuo, W. P., Jenssen, T. K., Butte, A. J., Ohno-Machado, L., & Kohane, I. S. (2002). Analysis of matched mRNA measurements from two different microarray technologies. *Bioinformatics*, *18*(3), 405–412.

Melucci, M. (2007). On rank correlation in information retrieval evaluation. *SIGIR Forum*, *41*(1), 18–33.

Morishita, S., & Sese, J. (2000). Traversing itemset lattice with statistical metric pruning. In *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems* (pp. 226–236). May 15–17, 2000, Dallas, Texas, USA. New York: Assoc. Comput. Math.

Pearson, K. (1920). Notes on the history of correlation. *Biometrika*, *13*, 25–45.

Reynolds, H. T. (1977). *The analysis of cross-classifications*. New York: Free Press.

Uno, T., Kiyomi, M., & Arimura, H. (2005). LCM ver. 3: collaboration of array, bitmap and prefix tree for frequent itemset mining. In *OSDM'05: proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations* (pp. 77–86). New York: Assoc. Comput. Math.

Xiong, H., Shekhar, S., Tan, P. N., & Kumar, V. (2004). Exploiting a support-based upper bound of pearson's correlation coefficient for efficiently identifying strongly correlated pairs. In W. Kim, R. Kohavi, J. Gehrke, & W. DuMouchel (Eds.), *Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 334–343). Seattle, Washington, USA, August 22–25, 2004. New York: Assoc. Comput. Math.

Xiong, H., Shekhar, S., Tan, P. N., & Kumar, V. (2006). TAPER: A two-step approach for all-strong-pairs correlation query in large databases. *IEEE Transactions on Knowledge and Data Engineering*, *18*(4), 493–508.

Xiong, H., Zhou, W., Brodie, M., & Ma, S. (2008). Top-k ϕ correlation computation. *INFORMS Journal on Computing*, *20*(4), 539–552.

Yilmaz, E., Aslam, J. A., & Robertson, S. (2008). A new rank correlation coefficient for information retrieval. In S.H. Myaeng, D.W. Oard, F. Sebastiani, T.S. Chua, & M.K. Leong (Eds.), *Proceedings of the 31st annual international ACM SIGIR conference on research and development in information retrieval, SIGIR 2008* (pp. 587–594). Singapore, July 20–24, 2008.

Zhou, W., & Xiong, H. (2008). Volatile correlation computation: a checkpoint view. In Y. Li, B. Liu, & S. Sarawagi (Eds.), *Proceedings of the 14th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 848–856). Las Vegas, Nevada, USA, August 24–27, 2008. New York: Assoc. Comput. Math.