Volatile Correlation Computation: A Checkpoint View

Wenjun Zhou
MSIS Department
Rutgers, the State University of New Jersey
wjzhou@pegasus.rutgers.edu

Hui Xiong
MSIS Department
Rutgers, the State University of New Jersey
hxiong@rutgers.edu

ABSTRACT

Recent years have witnessed increased interest in computing strongly correlated pairs in very large databases. Most previous studies have been focused on static data sets. However, in real-world applications, input data are often dynamic and must continually be updated. With such large and growing data sets, new research efforts are expected to develop an incremental solution for correlation computing. Along this line, in this paper, we propose a CHECK-POINT algorithm that can efficiently incorporate new transactions for correlation computing as they become available. Specifically, we set a checkpoint to establish a computation buffer, which can help us determine an upper bound for the correlation. This checkpoint bound can be exploited to identify a list of candidate pairs, which will be maintained and computed for correlations as new transactions are added into the database. However, if the total number of new transactions is beyond the buffer size, a new upper bound is computed by the new checkpoint and a new list of candidate pairs is identified. Experimental results on real-world data sets show that CHECK-POINT can significantly reduce the correlation computing cost in dynamic data sets and has the advantage of compacting the use of memory space.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*

General Terms

Algorithms

Keywords

Pearson's Correlation Coefficient, ϕ Correlation Coefficient, Volatile Correlation Computing, Checkpoint

1. INTRODUCTION

Given a set of data objects, the problem of correlation computing is concerned with identification of strongly-related (e.g. as measured by Pearson's correlation coefficient for pairs [13]) groups of data objects. Many important applications in science and business [2, 6, 12, 14] depend on efficient and effective correlation computing techniques to discover relationships within large collections of information. Despite the development of traditional statistical correlation computing techniques [4, 10, 8, 11, 9, 15, 16], researchers and practitioners are still facing increasing challenges to measure associations among data produced by emerging data-intensive applications.

Indeed, the size of real-world datasets is growing at an extraordinary rate, and these data are often dynamic and need to be continually updated. With such large and growing data sets, new research efforts are expected to develop an incremental solution for correlation computing. To that end, in this paper, we limit our scope to provide a pilot study of incrementally querying all item pairs with correlations above a user specified minimum correlation threshold when new data become available.

A straightforward approach is to recompute the correlations for all the item pairs every time that new data of transactions become available. However, for large data sets, this approach is infeasible, particularly if the application needs the results in a timely fashion. An alternative method is to use more space to save the time. Along this line, we describe 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 incremental correlation computing.

Specifically, we propose a CHECK-POINT algorithm that makes a time-space tradeoff and can efficiently incorporate new transactions for correlation computing as they become available. In the CHECK-POINT algorithm, we set a checkpoint to establish a computation buffer, which can help us to determine a correlation upper bound. This checkpoint bound can be exploited to identify a list of candidate pairs, which will be maintained and computed for correlations as new transactions are added into the database. However, if the total number of new transactions exceeds the buffer size, a new upper bound is computed by the new 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 do 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 by using slightly more time.

As demonstrated by our experimental results on several real-world data sets, CHECK-POINT can significantly reduce the computational cost compared to existing correlation computing benchmark algorithms, i.e. TAPER, in a dynamic data environment. Also, we observe that there is a trade-off between the use of space and the time by setting different checkpoint values. Indeed, the size of the candidate list increases with the increase of the checkpoint value. In contrast, the average computational savings is reduced with the increase of the checkpoint value. Finally, our experimental results show that CHECK-POINT, as compared to SAVE-ALL, can greatly reduce the use of memory space.

Overview. The remainder of this paper is organized as follows. In Section 2, we introduce some basic concepts and formulate the problem. Section 3 provides a checkpoint view for dynamic all-strong-pairs correlation queries. In Section 4, we describe the CHECK-POINT and SAVE-ALL algorithms. Section 5 shows the experimental results. Finally, in Section 6, we provide the concluding remarks.

2. PRELIMINARIES

In this section, we first introduce some basic concepts and notations that will be used in this paper. Then, we provide the problem formulation.

2.1 Basic Concepts

The ϕ **correlation coefficient** [13] is the computation form of the Pearson's correlation coefficient [5] for binary variables. In a 2×2 contingency table shown in Table 1, the calculation of the ϕ correlation coefficient reduces to

$$\phi = \frac{P_{(00)}P_{(11)} - P_{(01)}P_{(10)}}{\sqrt{P_{(0+)}P_{(1+)}P_{(+0)}P_{(+1)}}}, \qquad (1)$$

where $P_{(ij)}$, for $i \in \{0,1\}$ and $j \in \{0,1\}$, denotes the number of samples which are classified in the ith row and jth column of the table, and N is the total number of samples. Furthermore, we let $P_{(i+)}$ denote the total number of samples classified in the ith row, and we let $P_{(+j)}$ denote the total number of samples classified in the jth column. Thus, $P_{(i+)} = \sum_{j=0}^{1} P_{(ij)}$ *and* $P_{(+j)} = \sum_{i=0}^{1} P_{(ij)}$.

Table 1: A two-way contingency table of item A and item B.

		B		Row Total
		0	1	
A	0	$P_{(00)}$	$P_{(01)}$	$P_{(0+)}$
	1	$P_{(10)}$	$P_{(11)}$	$P_{(1+)}$
Column Total		$P_{(+0)}$	$P_{(+1)}$	N

Hence, when adopting the support measure of association rule mining [1], for two items a and b in a market basket database, we have $supp(a) = P_{(1+)}/N$, $supp(b) = P_{(+1)}/N$, and $supp(a,b) = P_{(11)}/N$. In Xiong et al. (2004) [15] the support form of the ϕ correlation coefficient has been derived, as shown in Equation 2.

$$\phi_{\{a,b\}} = \frac{supp(a,b) - supp(a)supp(b)}{\sqrt{supp(a)supp(b)(1-supp(a))(1-supp(b))}} \qquad (2)$$

Xiong et al. (2004) has also identified an upper bound for $\phi_{\{a,b\}}$ [15]. Without loss of generality, if we assume that $supp(a) \geq supp(b)$, then an upper bound for $\phi_{\{a,b\}}$ is

$$upper(\phi_{\{a,b\}}) = \sqrt{\frac{supp(b)}{supp(a)}}\sqrt{\frac{1-supp(a)}{1-supp(b)}} \qquad (3)$$

For the purpose of simplicity, we denote N_a as the number of transactions in the database that contain item a, N_b as the number of those containing item b, and N_{ab} as the number of those containing both items. Then $supp(a) = N_a/N$, $supp(b) = N_b/N$, and $supp(a,b) = N_{ab}/N$. Substituting into Equation 2, we can calculate the ϕ correlation coefficient for items a and b as

$$\phi_{\{a,b\}} = \frac{NN_{ab} - N_aN_b}{\sqrt{N_a(N-N_a)N_b(N-N_b)}}. \qquad (4)$$

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 correlation query problem [15]. In this paper, we investigate the all-strong-pairs correlation query problem in a dynamic data environment.

Specifically, every time a data set of S new transactions is added into the original database $\mathcal{D}$, we want to have the dynamically updated results from the all-strong-pairs correlation query. In other words, this all-strong-pairs correlation query can be a frequent task in a dynamic data environment. As a result, our goal is to develop an incremental, practical, and computation-efficient solution to this all-strong-pairs correlation query problem.

3. A CHECKPOINT VIEW

In this section, we first introduce the checkpoint principle. Then, we provide some theoretical foundations for the checkpoint framework.

3.1 The Checkpoint Principle

In general, there are three ways for developing the incremental solutions for frequent and dynamic all-strong-pairs correlation queries.

First, the simplest way is to recompute the correlation values for all the item pairs every time new data sets of transactions become available. Along this line, we can use an efficient static all-strong-pairs correlation query algorithm, such as TAPER [15], for each computation. However, for very large data sets, this approach is infeasible if data updates are very frequent and the application needs the result in a timely manner.

The second way is to use more space to save time [3]. 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 all-strong-pairs 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 needs 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.

Finally, we look for an answer 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 a correlation upper bound. Specifically, at a checkpoint, assuming that we know that ΔN ($\Delta N << N$) new transactions will be added into the database before the next checkpoint, we can develop an upper bound for all the item pairs on $N + \Delta N$ transactions. This upper bound has taken the newly added ΔN 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 all-strong-pairs correlation queries. While new transactions can be added into the buffer dynamically, we only need to maintain the intermediate results for item pairs in this candidate list as long as the cumulative number of new transactions is less than ΔN. The above process is illustrated in Figure 1.

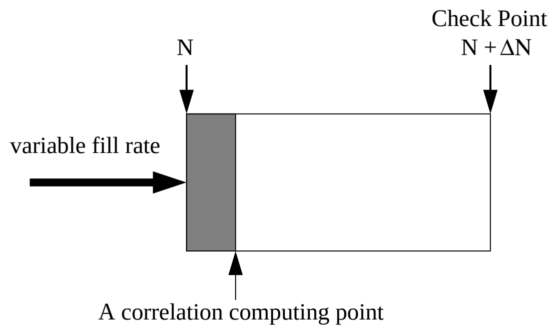


Figure 1: An Illustrate of the Checkpoint Process.

The reason that the candidate list can remain unchanged (as long as the cumulative number of new transactions is less than ΔN) is as follows. With a checkpoint at $N + \Delta N$, we identify upper bounds for all item pairs for all N known transactions and ΔN unknown transactions. In other words, these upper bounds are the maximum possible values they can achieve no matter what kind of ΔN transactions have been added into the database. Then, if the cumulative number of new transactions is less than ΔN, the upper bounds for all the item pairs in $N + \Delta N$ 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 ΔN, we need to set a new checkpoint at $N + 2\Delta N$. This iterative process will form an incremental solution for the dynamic all-strong pairs query problem. The rationale behind the checkpoint principle is that a small number of new transactions will not cause a significant effect on the correlation coefficients of most item pairs in the database if the total number of transactions is very large.

3.2 Predicted ϕ Correlation Coefficient at the Next Checkpoint

In Section 2.1 we have shown that the ϕ correlation coefficient can be computed by Equation 4. In the original database $\mathcal{D}$, the frequencies for item a, b, and item pair $\{a, b\}$ are denoted as N_a, N_b, and N_{ab}, respectively. Suppose that at a checkpoint, we set the next checkpoint right after ΔN new transactions. In the ΔN new transactions, we assume that there are ΔN_a, ΔN_b, and ΔN_{ab} new transactions containing item a, item b, and item pair $\{a, b\}$, respectively. Then according to Equation 4, at the next checkpoint the new ϕ correlation will be

$$\phi'_{\{a,b\}} = \frac{N'N'_{ab} - N'_aN'_b}{\sqrt{N'_a(N' - N'_a)N'_b(N' - N'_b)}}, \tag{5}$$

where$N' = N + \Delta N$, $N'_a = N_a + \Delta N_a$, $N'_b = N_b + \Delta N_b$, and $N'_{ab} = N_{ab} + \Delta N_{ab}$.

Unfortunately, we do not have any information about the ΔN new transactions, so ΔN_a, ΔN_b and ΔN_{ab} are all unknown. Thus, we cannot compute the new ϕ' directly. However, because the size of the new data set is much smaller than that of the original database, for any pair of items, the new ϕ' is expected not to change greatly from the current ϕ value. For this reason, we aim to derive an upper bound for ϕ', and use it as a criterion regarding whether we should save the intermediate computation result for that pairs. Specifically, if $upper(\phi')$ is less than the threshold θ, then we can guarantee that item pair $\{a, b\}$ will never become a strongly-correlated pair prior to the next checkpoint. As a result, we can save intermediate computation results only for pairs having ϕ' beyond the threshold θ.

However it is difficult to derive an exact upper bound for ϕ', because the denominator and the numerator are both affected by common factors, and they do not change monotonically by these factors. In the following subsections, we derive a *loose* upper bound for $\phi'_{\{a,b\}}$ when ΔN new transactions are added into the databases.

Looking closely at Equation 5, we can split the right hand side into three parts. Let $u = N'N'_{ab} - N'_aN'_b$, $v = N'_a(N' - N'_a)$, and $w = N'_b(N' - N'_b)$, then Equation 5 can be rewritten as

$$\phi'_{\{a,b\}} = \frac{u}{\sqrt{vw}}. \tag{6}$$

The upper bound is calculated as the maximum possible value of u divided by the product of the minimum possible values of v and w. The ratio is a loose upper bound for $\phi_{\{a,b\}}$ because the maximum and the minimum values may not be achieved simultaneously. Since we do not know the exact value of ΔN_a, ΔN_b, and ΔN_{ab} to come, our derivations are only based on the fact that $0 \leq \Delta N_{ab} \leq \Delta N_a \leq \Delta N$, and $0 \leq \Delta N_{ab} \leq \Delta N_b \leq \Delta N$.

3.3 Maximum Value of the Numerator u

In this subsection, we derive the maximum possible value for the numerator ϕ', u.

Given N, N_a, N_b, N_{ab}, and ΔN, the numerator of $\phi'_{\{a,b\}}$, written as $u = N'N'_{ab} - N'_aN'_b = (N + \Delta N)(N_{ab} + \Delta N_{ab}) -$

$(N_a + \Delta N_a)(N_b + \Delta N_b)$, is large when N_{ab} is large, and ΔN_a and ΔN_b are small. Specifically, we have the following lemma.

LEMMA 1. *Given N, N_a, N_b, N_{ab}, and ΔN, the maximum possible value for u, the numerator of the ϕ correlation coefficient $\phi_{\{a,b\}}$ at the next checkpoint, is*

$$u_{max} = \begin{cases} f(0) & \text{if } t \leq -\Delta N; \\ f(x^*) & \text{if } -\Delta N \leq t \leq \Delta N; \\ f(\Delta N) & \text{if } t \geq \Delta N, \end{cases} \tag{7}$$

where $f(x) = (N + \Delta N)(N_{ab} + x) - (N_a + x)(N_b + x)$, $t = N - N_a - N_b$, and $x^* = (N - N_a - N_b + \Delta N)/2$.

PROOF. Because of symmetry, we can assume without loss of generality that $\Delta N_a \leq \Delta N_b$. Let $\Delta N_{ab} = x$, $x \geq 0$; $\Delta N_a = x + c_1$, $c_1 \geq 0$; $\Delta N_b = x + c_1 + c_2$, $c_2 \geq 0$. In the following we derive the maximum value for u by taking first and second partial derivatives [7].

First, because $\partial u/\partial c_2 = -(N_a + x + c_1) < 0$, u increases monotonically as c_2 decreases. In order to reach the maximum of u, c_2 must take the minimum value in its range, 0. Similarly, because $\partial u/\partial c_1 = -(N_b + x + c_1 + c_2) - (N_a + x + c_1) < 0$, u takes the maximum value when $c_1 = 0$.

As a result, $\partial u/\partial x = (N + \Delta N) - (N_b + x + c_1 + c_2) - (N_a + x + c_1) = N - N_a - N_b + \Delta N - 2x$. Since $\partial^2 u/\partial x^2 = -2 < 0$, u reaches its maximum value when $\partial u/\partial x = 0$. Let $\partial u/\partial x = 0$, then the solution of the equation is $x^* = (N + \Delta N - N_a - N_b - 2c_1 - c_2)/2 = (N - N_a - N_b + \Delta N)/2$.

However, because $0 \leq x = \Delta N_{ab} \leq \Delta N$, the above value can be reached only if $-\Delta N \leq N - N_a - N_b \leq \Delta N$. If $N - N_a - N_b \leq -\Delta N$, then $\partial u/\partial x = N - N_a - N_b + \Delta N - 2x \leq -\Delta N + \Delta N - 2x = -2x \leq 0$, therefore u reaches its maximum when x takes its minimum possible value 0. On the other hand, if $N - N_a - N_b \geq \Delta N$, then $\partial u/\partial x = N - N_a - N_b + \Delta N - 2x \geq \Delta N + \Delta N - 2x = 2(\Delta N - x) \geq 0$. u reaches its maximum when x takes the maximum value ΔN. Now we have completed the proof of Lemma 1. □

3.4 Minimum Value of the Denominator $\sqrt{vw}$

In this subsection, we derive the minimum value of the denominator of ϕ', $\sqrt{vw}$. This is equivalent to finding the minimum value of vw.

First, a lower bound of vw can be reached by taking the minimum value of v and the minimum value of w. Again, the minimum values of v and w may or may not be reached simultaneously, so the lower bound for the denominator we derive here is also a loose bound.

LEMMA 2. *Given N, N_a, and ΔN, the minimum possible value for v in Equation 6 is*

$$v_{min} = \min\{h(N_a), h(N_a + \Delta N)\} \tag{8}$$

where $h(x) = x(N + \Delta N - x)$ is a function with respect to x, defined on the range $[0, N + \Delta N]$.

PROOF. Since $v = N'_a(N' - N'_a) = (N_a + \Delta N_a)(N + \Delta N - N_a - \Delta N_a) = h(N_a + \Delta N_a)$, finding the minimum of v is equivalent to finding the minimum value of function $h(x)$ within the range of $N_a + \Delta N_a$. Since $0 \leq \Delta N_a \leq \Delta N$, we have $N_a \leq N_a + \Delta N_a \leq N_a + \Delta N$. Now we will prove that the minimum possible value of $h(x)$ in the range $[N_a, N_a + \Delta N]$ is either $h(N_a)$ or $h(N_a + \Delta N)$.

Obviously $h(x)$ is a quadratic function of x. It is concave and symmetric with respect to $x = (N + \Delta N)/2$. Thus, the further $x = N_a + \Delta N_a$ is from $(N + \Delta N)/2$, the smaller $f(x)$ is. Because $N_a \geq 0$ and $N_a + \Delta N \leq N + \Delta N$, $N_a + \Delta N_a \in [N_a, N_a + \Delta N] \subseteq [0, N + \Delta N]$. The minimum value of $f(N_a + \Delta N_a)$ must be at some end of the range $[N_a, N_a + \Delta N]$, depending on whether N_a or $N_a + \Delta N$ is further from $(N + \Delta N)/2$. Now Lemma 2 has been proven. □

LEMMA 3. *Given N, N_b, and ΔN, the minimum possible value for w in Equation 6 is*

$$w_{min} = \min\{h(N_b), h(N_b + \Delta N)\} \tag{9}$$

where $h(x) = x(N + \Delta N - x)$ is a function with respect to x, defined on the range $[0, N + \Delta N]$.

PROOF. Similar to the proof of Lemma 2, we can simply prove that the minimum possible value of $h(x)$ in the range $[N_b, N_b + \Delta N]$ is either $h(N_b)$ or $h(N_b + \Delta N)$. □

3.5 The Loose Upper Bound for ϕ'

In this section, we can derive a loose upper bound for ϕ correlation coefficient $\phi = u/\sqrt{vw}$ by Lemma 1, Lemma 2, and Lemma 3 as the following:

LEMMA 4. *A loose upper bound for the new ϕ correlation coefficient at the next checkpoint, $\phi' = u/\sqrt{vw}$, is*

$$upper(\phi') = \frac{u_{max}}{\sqrt{v_{min} w_{min}}}, \tag{10}$$

where u_{max} follows Equation (7), v_{min} follows Equation (8), and w_{min} follows Equation (9).

PROOF. The proof is straightforward, since we have u_{max}, v_{min}, and w_{min} from Lemmas 1, 2, and 3. □

4. ALGORITHM DESCRIPTIONS

In this section, we describe three different solutions to the all-strong-pairs correlation query problem in a dynamic data environment. First, we present a straightforward solution, named r-TAPER, which recomputes correlation coefficients for all the item pairs every time new data are added into the database. The second SAVE-ALL approach stores the intermediate computing results for all the item pairs and can greatly save the computational cost. Finally, we also provide an incremental solution, called CHECK-POINT, which strikes a balance between the use of memory space and the computational efficiency.

4.1 The r-TAPER Algorithm

In this method, we need to recompute correlation coefficients for all the item pairs every time the database has been updated. No intermediate result has been reused for the next correlation computing practice. Since we have already known that a brute-force way to compute all-strong-pairs correlation queries is computationally expensive [15], we apply the TAPER algorithm [15] in this study. TAPER is an efficient algorithm for the all-strong-pairs correlation query on static data [15]. Figure 2 shows the pseudo code of the r-TAPER algorithm which computes all-strong-pairs queries in a dynamic data environment. In the figure, we can see that the r-TAPER algorithm repeatedly calls the TAPER procedure every time new transactions are added into the database. Note that the implementation details of TAPER can be found in [15].

```
ALGORITHM r-TAPER($D$,$D_\triangle$,$\theta$)
Input:
        $D$: the original database
        $D_\triangle$: the set of new transactions
        $\theta$: the threshold for pair-wise $\phi$ correlation
Output:
        $L$: list of item pairs with $\phi \geq \theta$.

1.      $D \leftarrow D \cup D_\triangle$
2.      $L \leftarrow TAPER(D, \theta)$
3.      Output $L$
```

Figure 2: The r-TAPER Algorithm

4.2 The SAVE-ALL Algorithm

In the r-TAPER algorithm, we have observed the fact that the computational bottleneck for the all-strong-pairs query is to count the frequencies of all item pairs on the fly, and no intermediate computing results have been reused for the next correlation computation. On the contrary, the SAVE-ALL method stores the frequencies of all item pairs and incrementally update the stored values while new data are added into the database. In this way, SAVE-ALL uses more space for the sake of saving computation time.

Figure 3 shows the implementation details of the SAVE-ALL algorithm. In this figure, Lines 1-5 process the dynamic data, and update frequencies of items and item pairs as needed. The frequencies of individual items are saved on the main diagonal. Lines 6-12 compute the ϕ correlation coefficient for each item pair and check if it is beyond the threshold θ. Since the frequencies of all the item pairs are stored in the memory, the computation of ϕ correlation coefficient for each item pair is very efficient. However, the drawback of this SAVE-ALL method is that, if the number of items becomes extremely large, we may not have enough memory space for running the algorithm.

```
ALGORITHM SAVE-ALL($M$,$D_\triangle$,$\theta$)
Input:
        $M$: $M[i,j]$ is the frequency of item pair $\{i,j\}$
        $D_\triangle$: the set of new transactions
        $\theta$: the threshold for pair-wise correlation
Output:
        $L$: list of item pairs with $\phi \geq \theta$.

1.      for each transaction $t$ in $D_\triangle$ do
2.          for each pair of items $\{i,j\}$ in $t$ do
3.              $M[i,j] \leftarrow M[i,j] + 1$
4.          end for
5.      end for
6.      $L \leftarrow \emptyset$
7.      for each possible pair $\{a,b\}$ do
8.          Compute $\phi_{\{a,b\}}$
9.          if $\phi_{\{a,b\}} \geq \theta$ then
10.             Add pair $\{a,b\}$ to $L$
11.         end if
12.     end for
13.     Output $L$
```

Figure 3: The Pseudocode of SAVE-ALL

4.3 The CHECK-POINT Algorithm

The above mentioned two algorithms are quite straight forward. However, they represent two extreme cases of the all-strong-pairs correlation query problem in a dynamic data environment. The r-TAPER algorithm, which repeat the query every time new data become available, disregards the previously computed results, and thus wastes a lot of computation. 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. As we have discussed in Section 3, we look for a solution in between the above two methods. This solution should have the capabilities in storing some intermediate computing results to save the computation, and do not overuse the memory space. Based on what we have discovered in Section 3, we have developed a new algorithm called CHECK-POINT, which is described in Figure 4.

```
ALGORITHM CHECK-POINT($CL$,$D$,$D_\triangle$,$\theta$,$Check$)
Input:
        $CL$: the candidate list
        $D$: the original database
        $D_\triangle$: a set of new transactions
        $\theta$: threshold for pair-wise correlation
        $Check$: a boolean variable indicating whether there
            is a checkpoint at the end of this step
Output:
        $L$: list of item pairs with $\phi \geq \theta$.

1.      for each transaction $t$ in $D_\triangle$ do
2.          for each pair of items $\{i,j\}$ in $t$ do
3.              if $\{i,j\} \in CL$ then
4.                  $N_{ij} \leftarrow N_{ij} + 1$
5.              end if
6.          end for
7.      end for
8.      $L \leftarrow \emptyset$
9.      for each pair $\{a,b\}$ in $C$ do
10.         Compute $\phi_{\{a,b\}}$
11.         if $\phi_{\{a,b\}} \geq \theta$ then
12.             Add pair $\{a,b\}$ to $L$
13.         end if
14.     end for
15.     Output $L$
16.     if Check then
17.         $D \leftarrow D \cup D_\triangle$
18.         $CL \leftarrow$ UpdateCandidateList($D$,$\Delta N$)
19.     end if
```

Figure 4: The Pseudocode of the CHECK-POINT Algorithm

In the figure, we can see that the CHECK-POINT algorithm consists of three parts. The first part, Lines 1-7, reads the new data and only updates the frequencies of item pairs on the candidate list. The second part, Lines 8-15, computes the ϕ correlation for each candidate pair and outputs if the correlation coefficient exceeds the threshold. Note that it is safe to ignore all other item pairs which are not on the candidate list. This is guaranteed by the way that the candidate list is constructed (please refer to the checkpoint principle in Section 3). The last part, described in Lines 16 through 19, calls a sub-procedure *UpdateCandidateList*, only if there is a checkpoint scheduled at the end of this step. In other words, the cumulative number of new transactions is greater than the computation buffer ΔN.

The *UpdateCandidateList* sub-procedure, described in Figure 5, shows what happens at each checkpoint. Given that

```
ALGORITHM UpdateCandidateList(D,ΔN)
1.    $CL \leftarrow \emptyset$
2.    for each possible item pair $\{a, b\}$ do
3.        Calculate the upper bound of $\phi'_{\{a,b\}}$ with $\Delta N$
4.        if $upper(\phi'_{\{a,b\}}) \geq \theta$ then
5.            Add $\{a, b\}$ to $CL$, and store $N_{ab}$
6.        end if
7.    end for
8.    return $CL$
```

Figure 5: The Pseudocode of the UpdateCandidateList subprocedure in CHECK-POINT.

the original data set has been updated, and that the frequency of each item is readily available, for each item pair $\{a, b\}$, we can compute an upper bound of its future ϕ correlation coefficient if we know that the next checkpoint is scheduled after ΔN new transactions to come. If the upper bound is no less than the threshold, then the item pair is put into the candidate list; otherwise we know that the item pair will never have a correlation coefficient above the user-specified correlation threshold θ even if another ΔN new transactions are added into the database.

5. EXPERIMENTAL RESULTS

In this section, we present the experimental results to evaluate the performance of the CHECK-POINT algorithm. Specifically, we study: (1) the computational performance of CHECK-POINT compared with r-TAPER and SAVE-ALL algorithms; (2) the performance of the CHECK-POINT algorithm in terms of the use of space.

5.1 The Experimental Setup

Our experiments were conducted on three real-world data sets: `chess`, `connect`, and `pumsb`. The first two data sets, `chess` and `connect`, are from UCI Machine Learning Repository (http://archive.ics.uci.edu/ml/). The last data set, `pumsb`, which is often used as a benchmark data set for evaluating frequent pattern mining algorithms, is from FIMI (http://fimi.cs.helsinki.fi/data/).

Table 2: Basic Characteristics of the Data Sets.

Data set	# Items	# Transactions	Source
`chess`	75	3196	UCI Repository
`connect`	127	67557	UCI Repository
`pumsb`	2113	49046	FIMI

Table 2 summarizes basic characteristics of the data sets used in our experiments. In this paper, our goal is to incrementally perform the all-strong-pairs correlation queries in a dynamic data situation. To mimic this dynamic real-world scenario, we did the following preprocessing on these three benchmark data sets. First, we generated a base data set of 100000 transactions for each data set by random sampling with replacement from the corresponding original data set. To make the results more comparable across different methods, step sizes, and checkpoint densities, we fixed the number of new transactions as 6000, a 6% increment of the base data. Again, these 6000 new transactions were generated by random sampling from the original data sets.

The difference between a step and a checkpoint is that a step is the number of new transactions after which we need an updated output of the strongly-correlated-pairs query, while a checkpoint is where we update the candidate list of item pairs. For example, an online store, which updates its bundle recommendations once they have received 1000 new transactions, has a step size 1000. The checkpoint size, however, is a parameter for the CHECK-POINT method, which the store can choose regarding how many transactions are to be collected between two neighboring checkpoints. Obviously, checkpoints do not apply to r-TAPER and SAVE-ALL, but these three methods can be compared with respect to the number of transactions processed.

For each data set, we carried out three groups of experiments. The first group aims to evaluate the effect of the checkpoint density for a fixed step size. Specifically, we fix the step size as 100 and use checkpoint sizes 200, 500, 1000, 1500, and 2000. The second group of experiments evaluates the effect of step sizes, in which we fix the checkpoint size as 1000, whereas try step sizes 10, 50, 100, 200, and 250. The last group of experiments fixes the ratio of checkpoint size versus step size. When using different step sizes, such as 10, 50, 100, 150, and 200, we insert a checkpoint at the end of every 10 steps, no matter what the step size is. The experimental groups are summarized in Tables 3, 4, and 5.

Table 3: Experimental Group I.

Step Size	Checkpoint Size	# Steps	# Checkpoints
100	200	2	30
100	500	5	12
100	1000	10	6
100	1500	15	4
100	2000	20	3

Table 4: Experimental Group II.

Step Size	Checkpoint Size	# Steps	# Checkpoints
10	1000	100	6
50	1000	20	6
100	1000	10	6
200	1000	5	6
250	1000	4	6

Table 5: Experimental Group III.

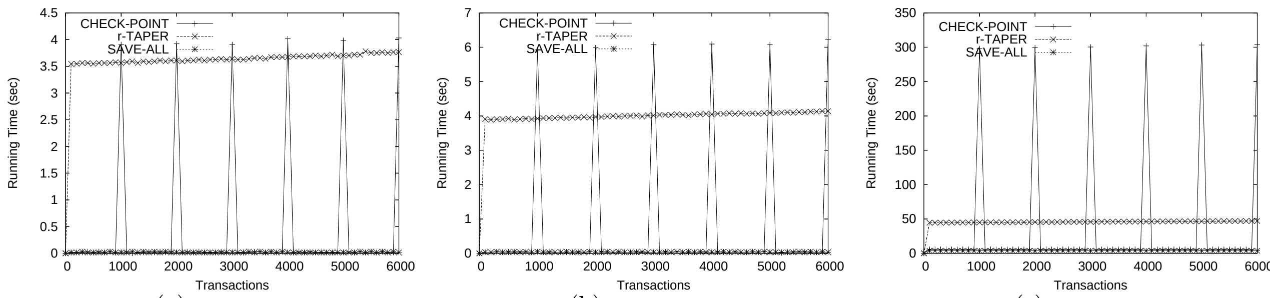
Step Size	Checkpoint Size	# Steps	# Checkpoints
10	100	10	60
50	500	10	12
100	1000	10	6
150	1500	10	4
200	2000	10	3

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

5.2 Computational Performance

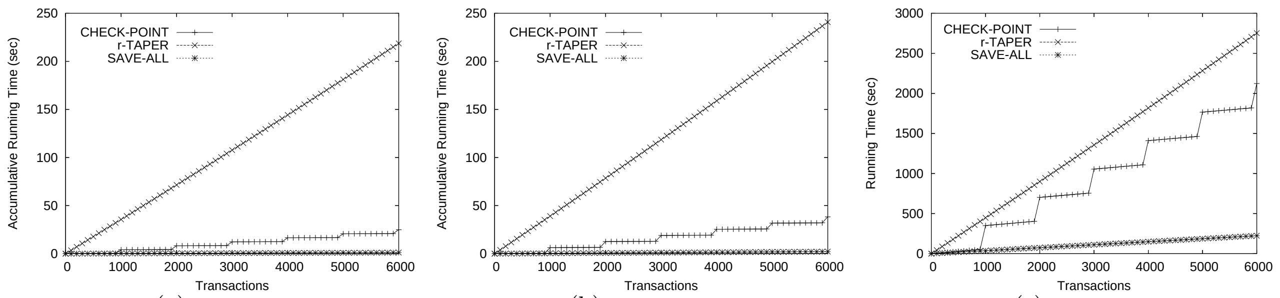
In this subsection, we show a comparison of computational performance for three algorithms: r-TAPER, SAVE-ALL, and CHECK-POINT.

Figure 6 illustrates the running time of each step for the `chess`, `connect`, and `pumsb` datasets. Unsurprisingly, the



(a) chess (b) connect (c) pumsb

Figure 6: The running time at each step of the CHECK-POINT, r-TAPER and SAVE-ALL algorithms on `chess`, `connect`, and `pumsb` datasets. ($S = 100$, $C = 1000$, $\theta = 0.5$).



(a) chess (b) connect (c) pumsb

Figure 7: Accumulative running time at each step of the CHECK-POINT, r-TAPER and SAVE-ALL algorithms on `chess`, `connect`, and `pumsb` datasets. ($S = 100$, $C = 1000$, $\theta = 0.5$).

SAVE-ALL algorithm costs the least time, because the pairwise frequencies are stored in the memory. r-TAPER takes much longer time than SAVE-ALL and CHECK-POINT, because all the pairwise frequencies are not available and need to be counted every step when new data become available. It is reasonable according to the r-TAPER algorithm that the higher the threshold, the more pairs are pruned and the less computation is needed. For different thresholds, the time consumed by the CHECK-POINT algorithm is almost always as little as SAVE-ALL, except for the running time at checkpoints, where CHECK-POINT need to take time for building a new candidate list of item pairs.

Even though CHECK-POINT takes longer time to update the candidate list at the checkpoints than one-running time of r-TAPER, overall the CHECK-POINT algorithm takes much less time than r-TAPER. In Figure 7, we illustrate the accumulative time at each step. Because the SAVE-ALL method is so fast, as time goes, its accumulative time grows very slowly. On the contrary, the accumulative time by r-TAPER increases much faster. The CHECK-POINT algorithm lies in between SAVE-ALL and r-TAPER. Its accumulative time increases slowly except for checkpoints where larger jumps can be observed. However, the overall trend shows that it increases much slower than r-TAPER.

Figure 8 shows a comparison of the CHECK-POINT, r-TAPER, and SAVE-ALL algorithms at different threshold levels. We have already known that the higher the thresholds, the more item pairs are pruned by the r-TAPER algorithm; however in the dynamically growing databases, as the threshold goes down, the running time of CHECK-POINT increases much more slowly than that of r-TAPER.

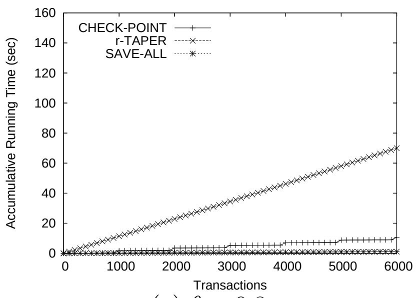
Figure 9 shows the effect of the checkpoint density on the running time. Because the checkpoints are the most costly steps, the more frequent we update the candidate list, the more time in total we will need. However, using too few checkpoints will require more space. We will show this tradeoff in the next subsection.

To study the effect of step sizes, we fix the checkpoint densities and plot the accumulative running time in Figure 10. We can see that all the curves are almost overlapping each other. Again, the reason is that the checkpoints are the most costly steps, thus the choice of step sizes does not affect the performance greatly. The implications on real world applications is that sizes of steps, where we want an update-to-date output of all the strongly correlated pairs, can be determined by the application itself. All we need to leverage is the choice of the checkpoint density, so that both the running time and the space required is reasonably balanced and practical.

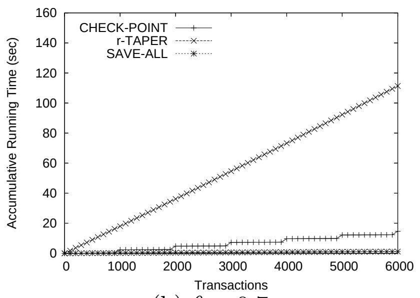
Finally, when fixing the step size and the checkpoint density, Figure 11 shows the effect of correlation thresholds on the running time. It is easy to see that, overall, the lower the threshold, the more running time is needed. Our experiments on other data sets and parameters show similar trends. Due to the page limit, we do not present these experimental results in this paper.

5.3 The Use of Space

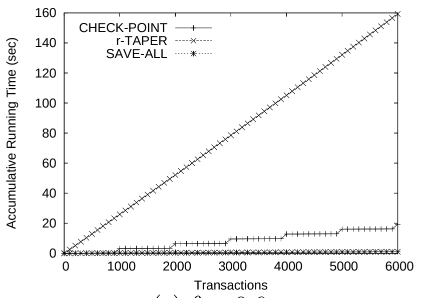
In this section, we investigate the performance of the CHECK-POINT algorithm in terms of the use of space. Along this line, our goal is to check how many item pairs can



(a) $\theta = 0.8$



(b) $\theta = 0.7$



(c) $\theta = 0.6$

Figure 8: Accumulative running time of the CHECK-POINT, r-TAPER and SAVE-ALL algorithms on chess ($S = 100$, $C = 1000$).

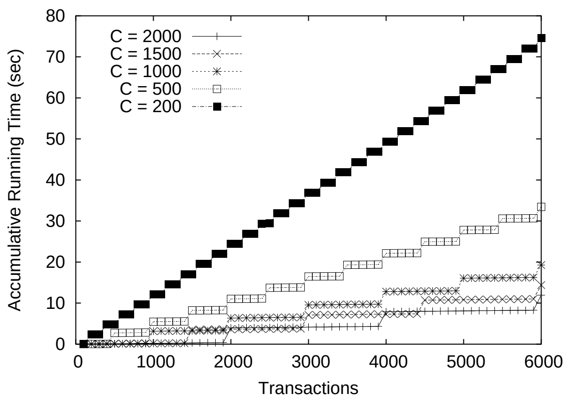


Figure 9: The accumulative running time at each step of the CHECK-POINT algorithm on chess. ($S = 100$, $\theta = 0.6$).

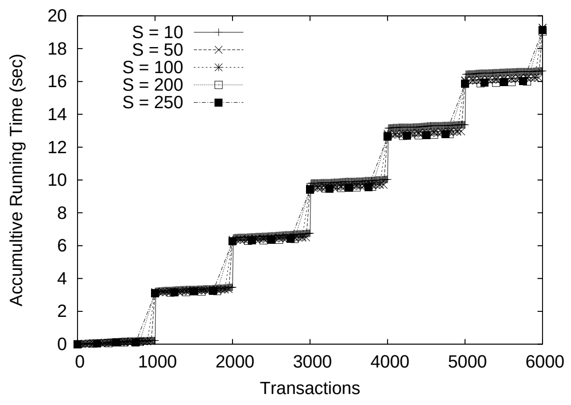


Figure 10: The accumulative running time at each step of the CHECK-POINT algorithm on chess. ($C = 1000$, $\theta = 0.6$).

be pruned at the checkpoints. In other words, these pruned item pairs will not be maintained in the candidate list

To study the pruning effect of different densities of checkpoints, we fix the step size and the correlation threshold, and then get the plots in Figure 12. In this figure, similar trends can be found for different data sets.

First of all, the denser the checkpoints, the fewer candidates are needed. In each of the subgraphs, each data point corresponds to a checkpoint. We can see that curves with fewer checkpoints lie higher than those with more checkpoints. This indicates that the less frequent the checkpoint is, the more candidates are needed to be stored.

Secondly, as time goes, the number of candidates may vary. Our experiments show that the number of candidates increases or decreases only slightly over time. The reason is that we generated the data sets uniformly at random, which eliminates the evolving trend over time in the original data sets. However, this may not be the case in practice.

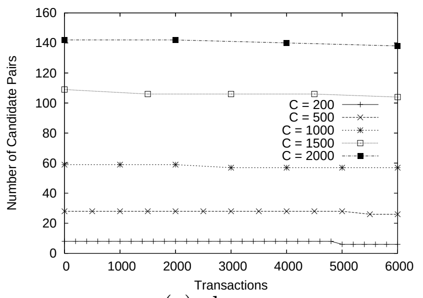
Finally, the pruning ratio is data dependent. As an example, Table 6 shows the sizes of candidate lists for three test data sets. In the table, we can see that that the number of candidate item pairs for chess is only 4, meaning that 99.86% of all possible item pairs are pruned. Instead of saving the frequencies of all 2775 item pairs, we only need to track the change of 4 pairs. Similarly, the connect data set has a pruning ratio of 97.60%. The pruning ratio for pumsb is only 45.57%, which explains why the computational performance of CHECK-POINT on the pumsb data set is not as good as on the other two data sets. The reason is that the chess and connect data sets are much denser than pumsb. Our proposed CHECK-POINT algorithm works better on dense data sets.

Table 6: The Sizes of Candidate Lists ($S = 10$, $C = 100$, $\theta = 0.7$).

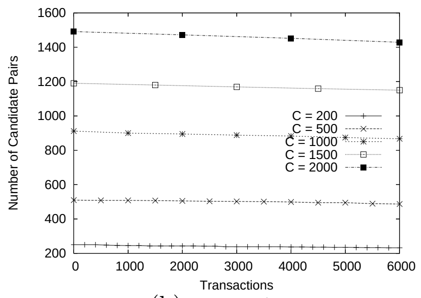
Data Set	#Items	#Pairs	#Cand's	Ratio
chess	75	2775	4	99.86%
connect	127	8001	191	97.60%
pumsb	2113	2231328	1214496	45.57%

6. CONCLUDING REMARKS

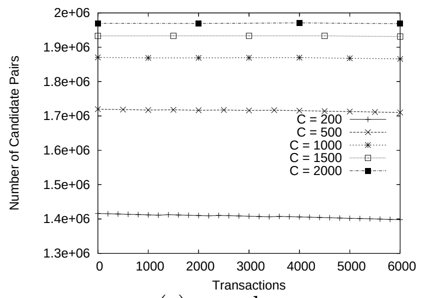
In this paper, we studied the problem of correlation computing in large and dynamically growing data sets. Specifically, we proposed a CHECK-POINT algorithm, which can incrementally search all the item pairs with correlations above a user-specified minimum correlation threshold. The key idea is to establish 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 computed for correlations as new transactions are added into the database. However, if the total number of new transactions is beyond the check point, a new candidate list is generated by the new checkpoint. Experimental results on real-world data sets show that CHECK-POINT



(a) chess



(b) connect



(c) pumsb

Figure 12: Number of candidate pairs at each checkpoint for chess, connect, and pumsb ($S = 100$, $\theta = 0.9$).

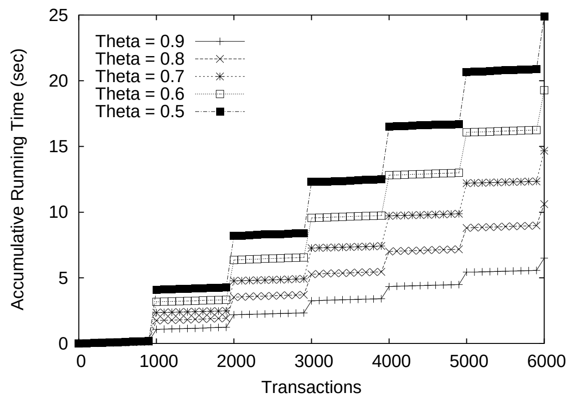


Figure 11: The accumulative running time at each step of the CHECK-POINT algorithm on chess. ($S = 100$, $C = 1000$).

can compact the use of memory space by maintaining a 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.

7. ACKNOWLEDGMENTS

This research was partially supported by the Rutgers Seed Funding for Collaborative Computing Research. Also, this research was supported in part by a Faculty Research Grant from Rutgers Business School- Newark and New Brunswick.

8. REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, 1993.

[2] C. Alexander. *Market Models: A Guide to Financial Data Analysis.* John Wiley & Sons, 2001.

[3] J. Bentley. *Programming Pearls.* Addison-Wesley, Inc., 2nd edition, 2000.

[4] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 265–276, 1997.

[5] J. Cohen, P. Cohen, S. West, and L. Aiken. *Applied multiple regression/correlation analysis for the behavioral sciences.* Lawrence Erlbaum Associates, Hillsdale, NJ, 3rd edition, 2003.

[6] P. Cohen, J. Cohen, S. G. West, and L. S. Aiken. *Applied Multiple Regression/Correlation Analysis for the Behavioral Science.* Lawrence Erlbaum Assoc; 3rd edition, 2002.

[7] R. Courant and F. John. *Introduction to Calculus and Analysis Volume II/1: Chapters 1 - 4 (Classics in Mathematics).* Springer, 1999.

[8] W. DuMouchel and D. Pregibon. Empirical bayes screening for multi-item associations. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 67–76, 2001.

[9] I. F. Ilyas, V. Markl, P. J. Haas, P. Brown, and A. Aboulnaga. Cords: Automatic discovery of correlations and soft functional dependencies. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, 2004.

[10] C. Jermaine. The computational complexity of high-dimensional correlation search. In *Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM)*, pages 249–256, 2001.

[11] C. Jermaine. Playing hide-and-seek with correlations. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.

[12] W. Kuo, T. Jenssen, A. Butte, L. Ohno-Machado, and I. Kohane. Analysis of matched mrna measurements from two different microarray technologies. *Bioinformatics*, 18(3), 2002.

[13] H. T. Reynolds. *The Analysis of Cross-classifications.* The Free Press, New York, 1977.

[14] H. V. Storch and F. W. Zwiers. *Statistical Analysis in Climate Research.* Cambridge University Press; Reprint edition, February 2002.

[15] H. Xiong, S. Shekhar, P. Tan, and V. Kumar. Exploiting a support-based upper bound of pearson's correlation coefficient for efficiently identifying strongly correlated pairs. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 334–343, 2004.

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