Space Reduction for Contextual Transaction Trust Computation in E-Commerce and E-Service Environments

Haibin Zhang
Department of Computing
Macquarie University
Sydney, Australia
haibin.zhang@mq.edu.au

Yan Wang
Department of Computing
Macquarie University
Sydney, Australia
yan.wang@mq.edu.au

Jian Yang
Department of Computing
Macquarie University
Sydney, Australia
jian.yang@mq.edu.au

***Abstract*—In the literature, Contextual Transaction Trust computation (termed as *CTT computation*) is considered an effective approach to evaluate the trustworthiness of a seller. Specifically, it computes a seller's reputation profile to indicate his/her dynamic trustworthiness in different product categories, price ranges, time periods, and any necessary combination of them. Then, in order to promptly answer a buyer's requests on the results of CTT computation, *CMK-tree* has been designed to appropriately index the precomputed aggregation results over large-scale ratings and transaction data. Nevertheless, *CMK-tree* requires additional storage space. In practice, a seller usually has a large volume of transactions. Moreover, with significant increase of historical transaction data (e.g., over one or two years), the size of storage space consumed by *CMK-tree* will become much larger.**

In reducing storage space consumption for CTT computation, the aggregation results that are generated based on the ratings and transaction data from remote history, e.g., "*12 months ago*" can be deleted, as the ratings from remote history are less important for evaluating a seller's recent behavior. However, to achieve nearly linear and robust query performance, the deletion operations in the *CMK-tree* become complicated. In this paper, we propose three deletion strategies for CTT computation based on *CMK-tree*. With our proposed deletion strategies, the additional storage space consumption can be restricted to a limited range, which offers great benefit to trust management with millions of sellers. Finally, we have conducted experiments to illustrate both advantages and disadvantages of the proposed deletion strategies.

***Keywords*-E-Commerce; Trust and Reputation; Contextual Transaction Trust; Deletion Strategy; Aggregation Index**

I. Introduction

With the rapid development of e-commerce and e-service, the trustworthiness of a seller or service provider is critically important to potential consumers in decision-making for a purchase [5]. Over the past few years, it has been receiving much attention from researchers to build various trust evaluation models [12], [16], [6]. Briefly speaking, the idea of most existing trust evaluation models is to rate sellers or service providers, and then use the aggregated ratings as the indication of their trustworthiness or reputation score. However, the above single value only reflects a seller's general trust status. With such trust evaluation models, consumers are vulnerable to some frauds from malicious sellers. For example, a malicious seller can gain a good reputation by honestly selling good and low value (price) products. Once having accumulated a good reputation, s/he may deceive buyers by inducing them to buy more expensive products, but either not delivering the ordered product or else delivering a fake product [10]. In the literature, this is referred to as the *value imbalance* problem [4].

Actually, the reason that most existing trust evaluation models cannot identify *value imbalance* in transactions is that they do not take transaction context into account [15]. In e-commerce environments, different transactions generally have different *natures* and *contexts*; even the same seller needs to be considered differently with regard to the trustworthiness in different forthcoming transactions [9], [15]. In addition, the single value computed by these trust evaluation models is static with regard to any forthcoming transaction of selling different products. As a result, they can hardly predict the likelihood of a successful forthcoming transaction [14], [22]. Note that e-commerce is also one type of e-service applications. As the *value imbalance* exists in both e-commerce and e-service environments, in the context of this paper, we do not explicitly differentiate them but take e-commerce as an example in analysis.

A. Background

In the literature, in contrast to most existing trust management models, Zhang *et al.* [23] have proposed a trust vector based framework, which has been demonstrated to be an effective approach to resolve the aforementioned problems. The trust vector consists of three major elements, which are called Contextual Transaction Trust (CTT) values. They are

1) ***Transaction Item Specific Trust (TIST):*** *the trustworthiness of a seller in selling a specific product to be traded in a forthcoming transaction;*
2) ***Product Category based Trust (PCT):*** *the trustworthiness of the seller in a layer in the product category hierarchy that is higher than the specific product to be traded in the forthcoming transaction, within a price range and a time period;*
3) ***Similar Transaction Amount based Trust (STAT):*** *the trustworthiness of the seller in a valid price range and a time period.*

For each element in the trust vector, the higher the value is, the more trustworthy the seller will be. When computing the last two elements, the parameters, such as product category, price range and time range, can be specified and adjusted by the buyer. For example, if "*Apple iPhone6 16GB*" at the price of around "*$700*" is the product in a forthcoming transaction, the buyer can specify and adjust "*product category*" along a path in the product category hierarchy, such as "*Apple iPhone*" and "*Smartphone*" in sequence. In the meantime, the buyer may also specify and adjust the price range and the time range. Each price range takes the price of a product as approximately the medium value, such as "*$500−$1000*". Each time range takes the recent time period, such as "*the latest 3 months*". They term the query on CTT values as a *CTT query*, and term the computation of CTT values as *CTT computation*. Hence, with all computed trust results, the reputation profile of a seller can be outlined, which can indicate the dynamic trustworthiness of a seller in different products and product categories, price ranges, time periods and necessary combination of them, greatly help identify the *value imbalance* problem potentially existing in forthcoming transactions.

In order to promptly answer a buyer's CTT queries on the dynamic trustworthiness of a seller, Zhang *et al.* give the preliminary solutions in [24]. Specifically, CTT computation is first modeled as an extended RA (Range Aggregate) problem [8] in data warehousing. Recently, an optimal solution *CMK-tree* has been introduced [25], and it is superior in efficiency of computing CTT values to the existing approaches.

B. New Challenge

In general, the approaches to CTT computation precompute the aggregates over historical large-scale ratings and transaction data of a seller, with all necessary combinations of three dimensions, i.e., *Product Category*, *Transaction Amount/Price* and *Transaction Time*. Then, the aggregation results are stored appropriately in the specialized index forming a tree structure. Here, the index containing some aggregates (see Section III-B) is referred to as the *aggregation index*. Nevertheless, with continuous growth in transaction time and significant increase of transaction data, the size of additional storage space for storing the aggregation index can become much larger.

To solve the above problem, we propose to delete index records that are generated based on ratings and transaction data from remote history[1] (e.g., "*12 months ago*"). In practice, the time range in CTT queries usually refers to the latest time period; therefore, the earlier ratings that are less important for evaluating a seller's recent behavior can be eliminated. More importantly, it restricts the additional storage space consumption to a limited range, which offers great benefit to trust management with millions of sellers.

As mentioned before, the *CMK-tree* [25] (a brief review on the *CMK-tree* is given in Section III) is considered as the most efficient approach to CTT computation. In particular, while answering a buyer's CTT queries, the *CMK-tree* has almost linear and robust query performance, i.e., query time is short and stable even if the query range has been enlarged. With the new transactions of a seller occurred everyday, their corresponding transaction records are continuously added to the database in chronological order. In the meantime, the *CMK-tree* will be updated accordingly. However, in Section IV, we will explain that, in order to achieve nearly linear and robust query performance, the deletion operations in the *CMK-tree* become complicated. Therefore, in this paper, we propose three deletion strategies for the *CMK-tree*. As a result, the *CMK-tree* can be more effectively applied to large-scale e-commerce websites in terms of efficiency and storage space consumption for CTT computation.

The paper is organized as follows. Section II reviews related work. Section III introduces the *CMK-tree*. The deletion strategies for CTT computation are given in Section IV. Section V evaluates our proposed strategies experimentally, and Section VI concludes our paper.

II. Related Work

A. Trust Evaluation

In the literature, the single-value trust valuation models have been proposed and applied in various application fields, including Peer-to-Peer (P2P) network [16], service-oriented computing (SOC) [6], e-commerce environments [12], etc. As stated before, these models do not take any contextual information into account. Consequently, if applied in e-commerce environments, they can hardly predict the likelihood of a successful forthcoming transaction.

In recent years, more studies pay attention to introducing transaction context in trust evaluation [16], [14], [22]; nevertheless, in-depth discussions and solutions are needed to focus on how to differentiate transaction contexts and take into account their impacts on trust evaluation. Rettinger *et al.* propose a context-sensitive trust evaluation model (IHRTM) using statistical relational learning [9]. However, a major disadvantage of IHRTM model is its high computational complexity that makes it difficult to be applied to large-scale e-commerce websites. Zhang *et al.* [25] improve related techniques in data warehousing to resolve CTT computation problem and proposed a *ReputationPro* trust model.

B. The Range Aggregate (RA) Problem

In a wider research literature, the CTT computation is somewhat similar to sales analysis from multiple perspectives in data warehouses (OLAP) and business intelligence [3]. In fact, according to the analysis introduced in [21], the RA (Range Aggregate) problem [8] is relatively close to CTT computation.

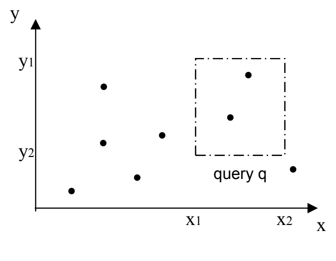

Figure 1. An example of an RA query

[1]Although storage space can be reduced to some extent by aggregating ratings and transaction data at different time granularity [20], it does not essentially resolve the problem of the large storage space consumption. Furthermore, it brings the problem of lowering accuracy of CTT results [25].

The traditional RA problem in a two-dimensional spatial data warehouse is to compute the total number of points that fall into a query region q surrounded by $[x_1, x_2]$ and $[y_1, y_2]$ (see Fig. 1). Some well-known disk-based index scheme, such as the *aR-tree* [8], the *aP-tree* [13], the *BA-tree* [18] and the *MVSB-tree* [19] are used for solving RA problem. However, they either have poor performance in computing CTT values or do not essentially meet the requirements of CTT computation. Therefore, Zhang *et al.* [25] propose a new index scheme the *CMK-tree*, which is particularly designed to efficiently support CTT computation.

C. The Multi-Version Structure

In the literature, the multi-version structure is an effective means to support aggregation operations along temporal dimension, e.g., *aP-tree* [13] and *MVSB-tree* [19]. A multi-version structure is to make *partial persistence*[2] a basic (tree) structure. For example, the *aP-tree* is an extended *Multi-Version B+-tree (MVBT)* [2] and the *MVSB-tree* is an *Multi-Version SB-tree* [17]. As stated before, the transaction records of a seller are added to database in chronological order which implies that the updates of generated index for CTT computation only affect the latest index records. Therefore, multi-version structure can also be adopted in the design of index schemes for CTT computation.

However, the multi-version structure does not support insertions to the historical (old) versions, which simultaneously limits its application, as the new points do not necessarily arrive in chronological order in some cases. To solve this problem, Tao *et al.* propose the *double logarithmic method* [13] which achieves both insertions and deletions to "history" for multi-version structure. In particular, for deletions, it takes advantage of an invertible deletion strategy [7]. In Section IV, we will apply the idea of the invertible deletion strategy and propose a *'naive' deletion strategy* for CTT computation. In addition, we introduce two other deletion strategies: the *space-effective deletion strategy* and the *time-effective deletion strategy*.

III. The CMK-tree: An Efficient Approach to CTT Computation

A. Three transaction context dimensions

In [23], Zhang *et al.* have identified three *context dimensions* with influence on the trustworthiness of a forthcoming transaction: *Product Category*, *Transaction Amount (Price)* and *Transaction Time*. In particular, the context of a transaction can be represented as different layers in the product category hierarchy and different ranges in each of the *Price* dimension and *Transaction Time* dimension.

Note that the *Product Category* dimension has a hierarchical structure. In *ReputationPro* trust model [25], a product category hierarchy with seven layers is used to support finer-grained analysis on contextual transaction trust with "*drill down*" and "*roll up*" operations in the hierarchy. Fig. 2 presents a small part of their extended product category

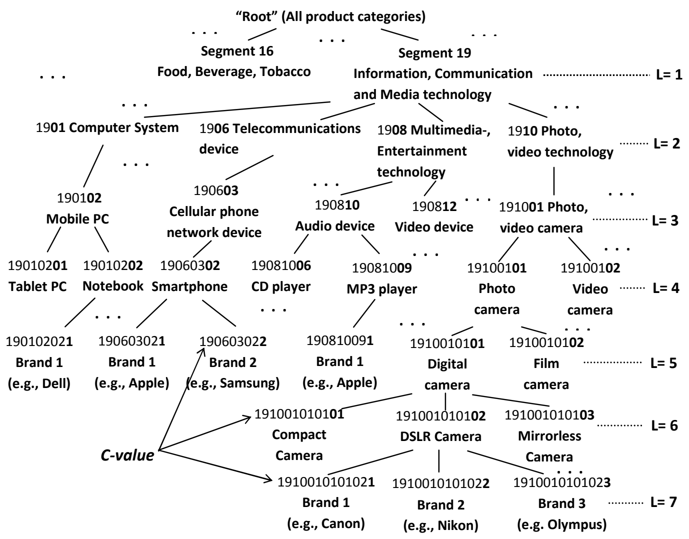

Figure 2. Part of the extended product category hierarchy of the segment *"Information, communication and media"*

hierarchy. Under each brand, there are corresponding products that belong to this brand.

B. CTT metrics

The existing studies average the rating values for calculating the trust value [6], [12], [16]. Following this idea, two aggregates are precomputed and stored separately. They are $count\_r$: the number of ratings of the corresponding transactions, and sum_r: the sum of ratings in a specific layer of product category hierarchy with a specific transaction price range and a specific time period. With a pair of $count\_r$ and sum_r, accordingly, the trust value can be computed as $T = \frac{sum\_r}{count\_r}$. In addition, based on the parameters of a CTT query, a set of $\{count\_r_i, sum\_r_i\}$ can be returned. Accordingly, the trust value is $T = \frac{\sum sum\_r_i}{\sum count\_r_i}$.

As pointed out by Zhang *et al.* [21], a CTT query can first be regarded as an RA problem in a two-dimensional space, where the x-axis represents the *Transaction Time* dimension in days and the y-axis represents the *Transaction Amount* dimension (see Fig. 1). Consequently, a CTT query on a seller in a time range $[t_1, t_2]$ and a transaction amount range $[ta_1, ta_2]$ can be converted by computing the number of the ratings $count\_r$ and the sum of the ratings sum_r of the transactions that fall into the query range formed by $[t_1, t_2]$ and $[ta_1, ta_2]$. Then, the Product Category is taken as the extended third dimension. Note that the above idea for CTT computation has been applied to design the index scheme proposed in [24], [25].

C. ReputationPro and CMK-tree

As introduced at the beginning of this paper, in view of the drawbacks of existing trust evaluation models, Zhang *et al.* present a trust vector consisting of three CTT values. In particular, with different parameters specified by a buyer regarding context dimensions, different sets of CTT values can be calculated to outline the reputation profile of the seller. They term this new model as *ReputationPro* [25].

[2]*partial persistence* [19] implies that updates are only applied to the latest version of the data structure, creating a linear ordering of versions.

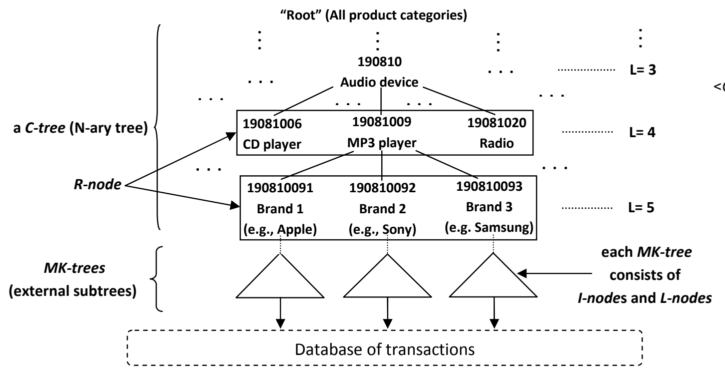

Figure 3. The structure of the CMK-tree

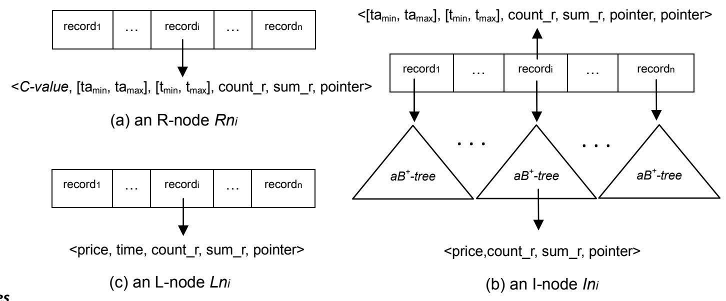
record1
…
recordi
…
recordn
<C-value, [tamin, tamax], [tmin, tmax], count_r, sum_r, pointer>
(a) an R-node Rni
<price, time, count_r, sum_r, pointer>
(c) an L-node Lni
<[tamin, tamax], [tmin, tmax], count_r, sum_r, pointer, pointer>
aB+-tree
<price,count_r, sum_r, pointer>
(b) an I-node Ini

Figure 5. The node structure of a CMK-tree

Moreover, an index scheme *CMK-tree* is proposed to solve the CTT computation problem. Generally speaking, as shown in Fig. 3, one *CMK-tree* consists of a *C-tree* and multiple *MK-trees* that are external to the *C-tree*. The *C-tree* consists of *R-node* (Rn), and an *MK-tree* consists of *I-node* (In) and *L-node* (Ln). Like *B/B*$^+$*-tree* [1], each of node can have multiple records depending on the node capacity.

1) The C-tree: Each record in an R-node maintains: a) *C-value* (see Fig. 2) to denote the unique id of the product category within the product category hierarchy; b) $[ta_{min}, ta_{max}]$ and $[t_{min}, t_{max}]$ to denote the transaction amount range and the transaction time range of all the transactions belonging to the current product category; c) $count\_r$ and sum_r to denote the aggregates over these transactions. Meanwhile, an R-node contains multiple product categories represented by corresponding records, and these product categories are on the same layer within the hierarchy (see Fig. 3). Clearly, all R-nodes form an *N-ary tree*, i.e. *C-tree (product* ***C****ategory-tree)*.

In addition, each record at a brand-based product category layer (e.g. '*Sony MP3 player*', see Fig. 3) is also a root of a subtree that is external to the *C-tree*. This subtree is similar to the original tree structures used for solving the RA problem, which records the aggregates $count\_r$ and sum_r in the corresponding *Transaction Amount* and *Transaction Time* dimensions.

2) The MK-tree: For each subtree that is external to the *C-tree*, the *MK-tree* is proposed based on extending the original *K-D-B-tree* [11]. In particular, it can be considered as an extended ***M****ulti-version "domain 0"* ***2****-D-B-tree*.

As depicted in Fig. 4(a), suppose that the x-axis represents the *Transaction Time* dimension in days and the y-axis represents the *Transaction Amount* dimension. In addition, each point in a two-dimensional space represents a transaction, and all the transactions belong to the same brand-based product category. Correspondingly, in an *MK-tree* generated by these transactions, a record in the I-node X surrounds the rectangular $a_1a_2a_3a_4$ (R_1, this is the first version of *"domain 0" 2-D-B-tree*). Another record in the I-node X surrounds the rectangular $b_1b_2b_3b_4$ (R_2) which is the second version. The record in the I-node Z at a higher level surrounds a larger rectangular space formed by $a_1a_2b_3b_4$. Furthermore, as plotted in Fig. 4(b), each transaction will generate an interval along the *Transaction Time* dimension. As an extended structure, the records in an I-node simultaneously maintain the transactions whose generated intervals intersect with the left border of the corresponding rectangle as well as the aggregates of transactions. Specifically, as shown in Fig. 5(b), multiple *aB*$^+$*-trees* are built to maintain the intersections between the generated intervals and the corresponding rectangles. Like *B*$^+$*-tree* [1], the transaction records in an *aB*$^+$*-tree* are kept sorted based on their transaction amount. Finally, each record appearing in an L-node points to the transaction record stored in the database. Fig. 5 shows the node structure of a *CMK-tree*. We close the review of the *CMK-tree* here. For its construction, details can be found in [25].

IV. The Deletion Strategies for CTT Computation

Next, let us consider the whole index structure of a *CMK-tree* where the multiple *MK-trees* are the major components. Zhang *et al.* [25] have proved that the *CMK-tree* has almost linear and robust query performance when answering a buyer's CTT queries for each brand-based product category, and this property largely depends on the well-designed *MK-tree*.

From the transformation plotted in Fig. 4(b), in order to achieve linear and robust query performance, each point[3] generates an interval along the *Transaction Time* dimension. Meanwhile, all versions of *"domain 0" 2-D-B-trees* index the points whose generated intervals intersect with the left border of them as well as aggregates of corresponding points. Thus, if the index records for points in a version of *"domain 0" 2-D-B-tree* are deleted, all its subsequent versions will be affected. For instance, in Fig. 4(b), while deleting the index record in *MK-tree* for the point α_1 in version 1 of *"domain 0" 2-D-B-tree*, the updates are also performed in version 2 and version 3, since their left borders are all intersected with intervals generated by the point α_1.

Property 1: The insertions in a *MK-tree* only affect the latest version of *"domain 0" 2-D-B-tree*, but the deletions in *MK-tree* affect all the subsequences of current version.

Clearly, due to Property 1, the deletions in the *CMK-tree* become quite complicated. Therefore, in the following, we will introduce three different deletion strategies for the *CMK-tree* so as to guarantee the generated aggregation index based

[3]In CTT computation, one point at (t_i, ta_i) may represent a set of repeated transactions that occurred on a day t_i selling the same product with the same price ta_i. Thus, we use the term *'point'* rather than *'transaction'* for accurate description.

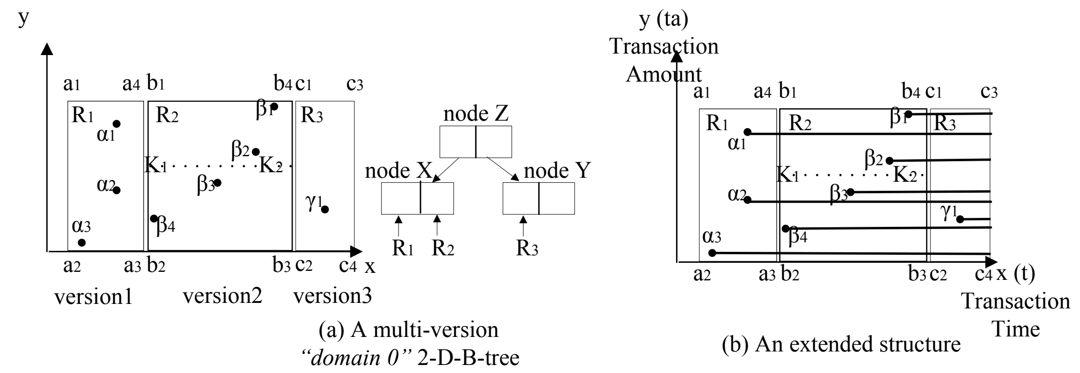

Figure 4. MK-tree – An extended multi-version "domain 0" 2-D-B-tree

on the ratings and transaction data within a specified time period, such as "*12 months*".

A. The 'Naive' Strategy

In [7], Overmars proposes a general invertible deletion strategy which periodically rebuilds the entire data structure. The main idea is that it has two insertion-only data structures, a *main* structure M and a *ghost* structure G. To insert an item, the insertion is performed in M. To delete an item, instead of removing it from M, the item is actually inserted to another structure G. Then, to avoid the size of M and G becoming much larger, the entire data structure is rebuilt periodically, i.e., building a new main structure M with the remaining items (the items are in M but not in G) and a new empty ghost structure G. The whole process is also termed *global rebuilding*.

As mentioned in Section II-C, Tao *et al.* propose a *double logarithmic method* [13] that adopts the idea of invertible deletion strategy to make an *aP-tree* fully dynamic, i.e., the *aP-tree* (a multi-version structure) can support insertions to the historical versions as well as the deletions. Here the discussion of the above dynamic data structure is beyond the scope of this paper, as the insertions are not performed in historical versions for CTT computation. In addition, we emphasize that the ghost structure is not needed for the 'naive' strategy. Instead, we only need to periodically rebuild the entire *CMK-tree* for each seller. To ease the overhead of rebuilding, the time period can be set to *a month* or even longer. In other words, for the 'naive' deletion strategy, the generated aggregation index may include "*12 months plus one month*, i.e., *13 months*" (taking "*12 months*" as the specified time period). The following two steps are conducted continuously for the 'naive' strategy.

Step 1: Perform insertions to the *CMK-tree* until all the transactions in past $m + n$ days have completed;

Step 2: Perform rebuilding the entire *CMK-tree* based on the ratings and transaction data of a seller in latest m days. Then, execution resumes from Step 1.

Summary: The idea of the 'naive' strategy is simple, and it avoids complex deletion operations introduced in Property 1. Furthermore, this strategy can be easily extended to make the *CMK-tree* fully dynamic [13]. However, it consumes more storage space for storing *CMK-tree* that is generated based on a longer time period. Also, periodically rebuilding it for each seller is time-consuming.

Different from the 'naive' strategy, the deletions are performed by *days* for the other two strategies. Specifically, in real operations, after all new transactions in the current day have completed and the *CMK-tree* is updated accordingly, the index records for the transactions completed on the first day, e.g., *"12 months ago"*, in the *CMK-tree* should be deleted so as to maintain the transaction information of a seller in a specific time period, e.g., *"12 months"*.

B. The Space-Effective Strategy

Basically, the space-effective strategy is to delete the index records in a *CMK-tree* directly. For the sake of simplicity, we take deleting the index record for the point α_3 as an example. Suppose that the point α_3 (see Fig. 4(b)) includes the information: *C-hrchy*$_{\alpha_3}$, ta_{α_3}, t_{first}, $count\_r_{\alpha_3}$ and $sum\_r_{\alpha_3}$ ($|count\_r_{\alpha_3}| \geq 1$, $|sum\_r_{\alpha_3}| \geq 1$). Note that *C-hrchy*$_{\alpha_3}$ represents the path in product category hierarchy (see Fig. 2) to which the point α_3 belongs. As the products traded in the transactions are at the bottom of product category hierarchy, the value of *C-hrchy*$_{\alpha_3}$ equals the *C-value* in the corresponding brand-based product category. t_{first} represents time of the first day, e.g., *"12 months ago"*. The following five steps are conducted for the space-effective strategy:

Step 1: A top-down (from R-nodes to L-nodes) search is performed in the *CMK-tree* to find the index record for the point α_3 based on the information of t_{first}, *C-hrchy*$_{\alpha_3}$, ta_{α_3}. Due to space constraints, we ignore that process for Step 1 and the detailed search process for locating an index record is given in [25].

Step 2: Remove the index record in an L-node for the point α_3. After removing the index record, the utilization of corresponding node may be quite low, i.e. the node contains only a few records. In Step 4, we will improve the node utilization via merging the sibling nodes.

Step 3: Update multiple *aB*$^+$*-trees*. In Property 1, we have explained that when the deletions are performed in an old version of *"domain 0" K-D-B-tree*, all its subsequent versions are affected. That is because the information of intersections between the generated intervals and the corresponding rectangles also have to be updated. As introduced in Section III-C2, multiple *aB*$^+$*-trees* are built to maintain the information of intersections; therefore, they should be updated accordingly, i.e. removing and merging records in the *aB*$^{+}$*-trees*. Note that removing and merging records in an *aB*$^{+}$*-tree* are the same as those of a *B*$^{+}$*-tree* [1].

Step 4: This step aims to improve the nodes utilization after removing some node records in Step 2. In [25], Zhang *et al.* have proved that the minimum space utilization is no less than $\frac{nc_L}{2}$ for an L-node in a *CMK-tree* where nc_L is the capacity of an L-node. Thus, if an L-node utilization is less than $\frac{nc_L}{2}$ after removing some records, the utilization of its sibling node will first be checked. During the process, if the sum utilization of current L-node and its sibling node is more than nc_L, these two nodes will not be merged. Otherwise, the records in these two L-nodes need to be merged. Meanwhile, update and merge the records in an I-node (as well as the ancestors of this I-node) which point to that two L-nodes.

Step 5: Update the ranges and aggregates from bottom up accordingly, i.e. update $[ta_{min}, ta_{max}]$, $[t_{min}, t_{max}]$, $count\_r$, sum_r maintained in corresponding R-nodes and L-nodes (see Fig. 5(a) and (b)). Here, in contrast to insertions where the *CMK-tree* is updated from top (R-node, the product category root) to bottom (L-node, pointing to the transaction record stored in the database), the updates are performed from bottom up for deletions.

Finally, the above five steps are performed in all the index records in the *CMK-tree* for the points with same transaction time t_{first}. In practice, in order to ease the overhead, Step 4 and Step 5 can be executed when all the index records for the transactions within a brand-based product category, which meet the requirements, have been deleted.

Summary: An important advantage of space-effective deletion strategy is that there is no extra storage space needed during deletion operations. However, inevitably, this strategy is still time-consuming, since it does not fundamentally resolve the problem caused by Property 1.

C. The Time-Effective Strategy

Generally speaking, both 'naive' deletion strategy and space-effective deletion strategy are time-consuming, as they do not solve the problem caused by Property 1. In particular, for 'naive' deletion strategy, the deletions themselves have been avoided; for space-effective deletion strategy, searching and updating the corresponding *aB*$^{+}$*-trees* in Step 3 are time-consuming. To solve this problem, a basic idea is to add indexes to manage the multiple *aB*$^{+}$*-trees* separately which can improve the performance of deletion operations.

Now, let us consider the structure of an I-node depicted in Fig. 5(b), where each record in an I-node maintains: $[ta_{min}, ta_{max}]$, $[t_{min}, t_{max}]$, $count\_r$, sum_r and two pointers. Note that one pointer points to an L-node, and the other points to an *aB*$^{+}$*-tree*. $t_{min} : [ta_{min}, ta_{max}]$ implies the left border of the rectangle surrounded by a record; $[ta_{min}, ta_{max}]$ is transaction amount range within which the *aB*$^{+}$*-tree* is built; $count\_r$ and sum_r are the aggregates of points (transactions) whose generated intervals intersect with $t_{min} : [ta_{min}, ta_{max}]$.

In the time-effective deletion strategy, we will design and add a new index to manage that multiple *aB*$^{+}$*-trees* pointed

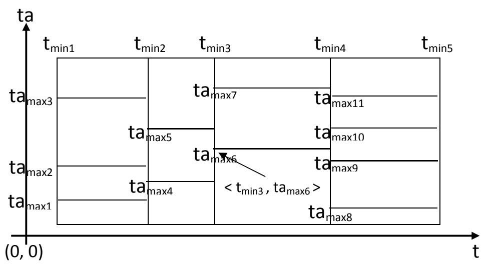

Figure 6. The 2-D matrix formed by the new index

by the I-nodes. Specifically, the index is built in following ways:

1) *The pair* $< t_{min}, ta_{max} >$ *in the record of an I-node is adopted as the unique ID to identify each aB*$^{+}$*-tree.* Therefore, each record in the new index has the form $< t_{min}, ta_{max}, pointer >$, where the pointer points to an *aB*$^{+}$*-tree*.

2) *The new index is built in each brand-based product category.* In a *CMK-tree*, the *aB*$^{+}$*-trees* are included in the *MK-trees* which are external to a *C-tree* (see Fig. 3). Thus, each record in an R-node at the brand-based product category layer, i.e. the bottom of the *C-tree*, simultaneously points to the new index.

3) *The new index can be organized to form a 2-D matrix.* Fig. 6 shows an example of the 2-D matrix. When searching the *aB-trees* that are effected by a deletion, they are easily located via this 2-D matrix. For instance, to delete the point α_3 with $< t_{first}, ta_{\alpha_3} >$ ($t_{first} < t_{min_1}$, $ta_{max4} < ta_{\alpha_3} < ta_{max2}$), the *aB-trees* with IDs $< t_{min1}, ta_{max2} >$, $< t_{min2}, ta_{max5} >$, $< t_{min3}, ta_{max6} >$, $< t_{min4}, ta_{max9} >$ are selected. Then, the corresponding *aB-trees* will be updated accordingly. Note that as the time increases, the 2-D matrix also needs to be updated.

Except for the Step 3 in space-effective strategy, the time-effective strategy has the same deletion process as space-effective strategy. However, after introducing the new index, the performance of deletion operations can be improved. Also, their performance will be demonstrated in the experiments introduced in Section V.

Summary: For the time-effective deletion strategy, the new index is added so as to reduce time-consuming for deletions in a *CMK-tree*. In this way, the information of intersections between the generated intervals and the corresponding rectangles (see Fig. 4(b)) will be managed separately. To some extent, this strategy can be regarded as a trade-off between time and storage space.

V. Experiments

In this section, we introduce the experiments conducted on four large-scale synthetic datasets generated from two real-world datasets, which aim to evaluate both advantage and disadvantage of three proposed deletion strategies.

A. Datasets

Firstly, with eBay APIs[4], we have obtained detailed feedback and transaction data for up to "*90 days*" of selected

[4]developer.ebay.com/support/docs

Table I
STORAGE SPACE CONSUMPTION FOR THREE DIFFERENT DELETION STRATEGIES BASED ON FOUR DATASETS

datasets	'naive' strategy	space-effective strategy	time-effective strategy
$SD_1(S_1)$	the size of *CMK-tree* built on $SD_1(S_1)'$: 0.89 MB the size of *CMK-tree* built on $SD_1(S_1)$: 0.82 MB extra space required: 0.07 MB (8.5% more)	no extra space required	extra space required: 40 KB (4.8% more than space-effective strategy)
$SD_2(S_1)$	the size of *CMK-tree* built on $SD_2(S_1)'$: 2.4 MB the size of *CMK-tree* built on $SD_2(S_1)$: 2.2 MB extra space required: 0.2 MB (9.0% more)	no extra space required	extra space required: 140 KB (6.0% more than space-effective strategy)
$SD_3(S_2)$	the size of *CMK-tree* built on $SD_3(S_2)'$: 0.28 MB the size of *CMK-tree* built on $SD_3(S_2)$: 0.26 MB extra space required: 0.02 MB (7.7% more)	no extra space required	extra space required: 10 KB (4.0% more than space-effective strategy)
$SD_4(S_2)$	the size of *CMK-tree* built on $SD_4(S_2)'$: 2.53 MB the size of *CMK-tree* built on $SD_4(S_2)$: 2.3 MB extra space required: 0.23 MB (10.0% more)	no extra space required	extra space required: 150 KB (6.4% more than space-effective strategy)

Table II
TIME CONSUMPTION OF THREE DIFFERENT DELETION STRATEGIES BASED ON FOUR DATASETS (I/O)

datasets	'naive' strategy	space-effective strategy	time-effective strategy
$SD_1(S_1)$	21 mins (the *CMK-tree* construction time)	4.5 secs	4.2 secs (6.7% less than space-effective strategy)
$SD_2(S_1)$	25 mins (the *CMK-tree* construction time)	15.6 secs	13.0 secs (16.7% less than space-effective strategy)
$SD_3(S_2)$	4.2 mins (the *CMK-tree* construction time)	1.5 secs	1.4 secs (6.7% less than space-effective strategy)
$SD_4(S_2)$	6.7 mins (the *CMK-tree* construction time)	19.0 secs	16.2 secs (14.7% less than space-effective strategy)

sellers. In seller selection, we first chose a number of popular products, and the sellers selling them with the largest number of reviews. With them, we finally selected two sellers S_1 and S_2 who had totally around 12,000 transactions (approx. 133 transactions per day) and 4,000 transactions (approx. 44 transactions per day), respectively. In addition, the products sold by S_1 and S_2 distribute in multiple product categories; but most products are in the category of '*Information, Communication and Media technology*' (see Fig. 2).

Secondly, considering that only "*90 days*" real transaction data of a seller can be obtained from eBay, and the time range in CTT computation can be *"the latest 12 months"*, we generated four large synthetic datasets $\mathbf{SD_1(S_1)}$, $\mathbf{SD_2(S_1)}$, $\mathbf{SD_3(S_2)}$ and $\mathbf{SD_4(S_2)}$ containing transaction data of "*12 months*" based on the above two eBay sellers.

(1) **Type I** includes $\mathbf{SD_1(S_1)}$ for S_1 and $\mathbf{SD_3(S_2)}$ for S_2: To generate the two datasets, we duplicated the transaction data of each seller 10 times on a given day, and then duplicated the transactions data of "*90 days*" to form the transaction data of "*12 months*". Thus, two datasets contain about $480k$ and $160k$ transactions in total (i.e., approx. 1330 transactions per day for S_1 and 440 transactions per day for S_2), respectively. Type I synthetic datasets guarantee that the proportion of each sold product is the same on a daily basis in both the synthetic dataset and the corresponding real dataset.

(2) **Type II** includes $\mathbf{SD_2(S_1)}$ for S_1 and $\mathbf{SD_4(S_2)}$ for S_2: Each dataset contains transaction data 10 times as much as that of the seller on a given day, and contains transaction data for "*12 months*" totally. In each Type II synthetic dataset, a transaction is randomly selected from the corresponding seller's eBay real dataset. In Type II synthetic datasets, any product appears in the corresponding real dataset. But the proportion of it on a day or a month is different to that in the real dataset.

Finally, in order to evaluate our proposed deletion strategies, we further extended four synthetic datasets by randomly selecting *a month* of transactions in each dataset as transactions of "*12 months ago*". In this way, we obtained another four datasets $\mathbf{SD_1(S_1)}'$, $\mathbf{SD_2(S_1)}'$, $\mathbf{SD_3(S_2)}'$ and $\mathbf{SD_4(S_2)}'$ containing transaction data of "*13 months*".

To sum up, the purpose of generating these synthetic datasets is to test the performance of three deletion strategies under the circumstance with a large volume of transactions in both each day and a long time period.

B. Results

The experiments compared the three deletion strategies from two aspects: (1) deletion time consumption and (2) storage space consumption. In addition, the *CMK-tree* as well as the deletion strategies are implemented using VC++ 6.0 running on a Lenovo Y560 laptop with an Intel Core i5 CPU (2.20GHz), 2GB RAM, Windows 7 Professional operation system and MySql 5.1.35 relational database.

Storage space consumption: Table I presents the additional storage space consumption of three deletion strategies based on four datasets, i.e. from $SD_1(S_1)$ to $SD_4(S_2)$. 1) For the 'naive' deletion strategy, for instance, the storage space consumption based on $SD_1(S_1)$ equals to the size of *CMK-tree* built on $SD_1(S_1)'$ (including "*13 months*" transaction data) subtracts the size of *CMK-tree* built on $SD_1(S_1)$ (including "*12 months*" transaction data), i.e., 0.07 MB = 0.89 MB - 0.82 MB. Overall, based on four datasets, it leads to 7.7%–10.0% additional storage space consumption (i.e., 0.02 MB–0.23 MB); 2) For the space-effective deletion strategy, as illustrated in Section IV-B, there is no extra storage space needed during deletion operations; 3) For the time-effective deletion strategy, the storage space consumption equals to the size of added new index. Overall, it leads to 4.0%–6.4% extra space consumption for constructing additional new indices (i.e., 10 KB–150 KB).

Time consumption of deletions: Table II presents time consumption of three deletion strategies based on four datasets, i.e. from $SD_1(S_1)$ to $SD_4(S_2)$. 1) For the 'naive' strategy, the deletion time is equivalent to the constructing time of the *CMK-tree* built on four datasets (including "*12 months*" transaction data). Overall, based on four datasets, it leads to 4.2 mins–25 mins for constructing the corresponding *CMK-tree*. For the other two strategies, we have tested the time of deleting the index records for the transactions completed on the first day, respectively. 2) For the space-effective strategy, the total deletion time is 1.5 secs–21.0 secs based on four datasets; 3) For the time-effective strategy, the total deletion time is 1.4 secs–15.2 secs based on four datasets. In all the cases, the time-effective strategy is faster by 6.7%–16.7% in deletion time than space-effective strategy. On average, it is 11.2% faster.

Summary: According to the above experimental results, we conclude that, compared with the other two strategies, the time-effective deletion strategy brings a small increase in the storage space with much gain in the improvement of time consumption in deletion operations.

VI. Conclusion

The existing work for Contextual Transaction Trust (CTT) computation adopts a well-designed index scheme [25], but it consumes a large storage space with significant increase of historical transaction data. To make the existing index scheme for CTT computation scalable to large-scale e-commerce websites, in this paper, we have proposed three different deletion strategies which aim to delete index records that are generated based on ratings and transaction data from "*12 months ago*". From the results of our experiments conducted on four synthetic datasets, we can draw a conclusion that the time-effective deletion strategy provides a good trade-off between time and storage space.

References

[1] R. Bayer and E. M. McCreight. Organization and maintenance of large ordered indices. *Acta Informatica*, 1:173–189, 1972.

[2] B. Becker, S. Gschwind, T. Ohler, B. Seeger, and P. Widmayer. An asymptotically optimal multiversion b-tree. *VLDB Journal*, 5(4):264–275, 1996.

[3] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *SIGMOD Record*, 26(1):65–74, 1997.

[4] A. Jøsang and J. Golbeck. Challenges for robust of trust and reputation systems. In *International Workshop on Security and Trust Management*, 2009.

[5] A. Jøsang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644, 2007.

[6] Z. Malik and A. Bouguettaya. Rateweb: Reputation assessment for trust establishment among web services. *VLDB Journal*, 18(4):885–911, 2009.

[7] M. H. Overmars. The design of dynamic data structures. *Lecture Notes in Computer Science*, 156, 1983.

[8] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient olap operations in spatial data warehouses. In *SSTD'2001*, pages 443–459.

[9] A. Rettinger, M. Nickles, and V. Tresp. Statistical relational learning of trust. *Machine learning*, 82(2):191–209, 2011.

[10] B. Rietjens. Trust and reputation on ebay: Towards a legal framework for feedback intermediaries. *Information and Communications Technology Law.*, 15(1):55–78, 2006.

[11] J. Robinson. The k-d-b-tree: A search structure for large multidimensional dynamic indexes. In *SIGMOD'1981*, pages 10–18.

[12] J. Sabater and C. Sierra. Regret: Reputation in gregarious societies. In *ACM AGENTS'2001*, pages 194–195.

[13] Y. Tao, J. Zhang, D. Papadias, and N. Mamoulis. Range aggregate processing in spatial databases. *IEEE Transactions on Knowledge and Data Engineering*, 16(12):1555–1570, 2004.

[14] Y. Wang and E.-P. Lim. The evaluation of situational transaction trust in e-service environments. In *ICEBE'2008*, pages 265–272.

[15] Y. Wang and K.-J. Lin. Reputation-oriented trustworthy computing in e-commerce environments. *IEEE Internet Computing*, 12(4):55–59, 2008.

[16] L. Xiong and L. Liu. Peertrust: Supporting reputation-based trust in peer-to-peer communities. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):843–857, 2004.

[17] J. Yang and J. Widom. Incremental computation and maintenance of temporal aggregates. *VLDB Journal*, 12(3):262–283, 2003.

[18] D. Zhang, A. Markowetz, V. J. Tsotras, D. Gunopulos, and B. Seeger. Efficient aggregation over objects with extent. In *SIGMOD/PODS'2002*, pages 121–132.

[19] D. Zhang, A. Markowetz, V. J. Tsotras, D. Gunopulos, and B. Seeger. On computing temporal aggregates with range predicates. *ACM Transactions on Database Systems*, 33(2):12:1–12:38, 2008.

[20] H. Zhang and Y. Wang. A novel model for contextual transaction trust computation with fixed storage space in e-commerce and e-service environments. In *SCC'2013*, pages 667–674.

[21] H. Zhang, Y. Wang, and X. Zhang. Efficient contextual transaction trust computation in e-commerce environments. In *TrustCom'2012*, pages 318–325.

[22] H. Zhang, Y. Wang, and X. Zhang. Transaction similarity-based contextual trust evaluation in e-commerce and e-service environments. In *ICWS'2011*, pages 500–507.

[23] H. Zhang, Y. Wang, and X. Zhang. A trust vector approach to transaction context-aware trust evaluation in e-commerce and e-service environments. In *SOCA'2012*, pages 1–8.

[24] H. Zhang, Y. Wang, and X. Zhang. The approaches to contextual transaction trust computation in e-commerce environments. *Security and Communication Networks*, 7(9):1331–1351, 2014.

[25] H. Zhang, Y. Wang, X. Zhang, and E.-P. Lim. Reputationpro: The efficient approaches to contextual transaction trust computation in e-commerce environments. *ACM Transactions on the Web*, 9(1):2:1–2:49, 2015.