A

ReputationPro: The Efficient Approaches to Contextual Transaction Trust Computation in E-Commerce Environments

HAIBIN ZHANG, Macquarie University, Sydney, Australia
YAN WANG, Macquarie University, Sydney, Australia
XIUZHEN ZHANG, RMIT University, Melbourne, Australia
EE-PENG LIM, Singapore Management University, Singapore

In e-commerce environments, the trustworthiness of a seller is utterly important to potential buyers, especially when a seller is not known to them. Most existing trust evaluation models compute a single value to reflect the general trustworthiness of a seller without taking any transaction context information into account. With such a result as the indication of reputation, a buyer may be easily deceived by a malicious seller in a transaction where the notorious *value imbalance* problem is involved, i.e., a malicious seller accumulates a high level reputation by selling cheap products then deceives buyers by inducing them to purchase more expensive products.

In this paper, we first present a trust vector consisting of three values for Contextual Transaction Trust (CTT). In the computation of CTT values, three identified important *context dimensions*, including Product Category, Transaction Amount and Transaction Time, are taken into account. In the meantime, the computation of each CTT value is based on both past transactions and the forthcoming transaction. In particular, with different parameters specified by a buyer regarding context dimensions, different sets of CTT values can be calculated. As a result, all these trust values can outline the reputation profile of a seller that indicates the dynamic trustworthiness of a seller in different products, product categories, price ranges, time periods, and any necessary combination of them. We name this new model *ReputationPro*. Nevertheless, in *ReputationPro*, the computation of reputation profile requires new data structures for appropriately indexing the pre-computation of aggregates over large-scale ratings and transaction data in three context dimensions, as well as novel algorithms for promptly answering buyers' CTT queries. In addition, storing pre-computed aggregation results consumes a large volume of space particularly for a system with millions of sellers. Therefore, reducing storage space for aggregation results is also a great demand.

To solve these challenging problems, we first propose a new index scheme *CMK-tree* by extending the two-dimensional *K-D-B-tree* that indexes spatial data to support efficient computation of CTT values. Then, we further extend the *CMK-tree* and propose a *CMK-tree*^{RS} approach to reducing the storage space allocated to each seller. The two approaches not only are applicable to three context dimensions that are either linear or hierarchical, but also take into account the characteristics of the transaction-time model, i.e., transaction data is inserted in chronological order. Moreover, the proposed data structures can index each specific product traded in a time period in order to compute the trustworthiness of a seller in selling a product. Finally, the experimental results illustrate that the *CMK-tree* is superior in efficiency of computing CTT values to all three existing approaches in the literature. In particular, while answering a buyer's CTT queries for each brand-based product category, the *CMK-tree* has almost linear query performance. In addition, with significantly reduced storage space, the *CMK-tree*^{RS} approach can further improve the efficiency in computing CTT values. Therefore, our proposed *ReputationPro* model is scalable to large-scale e-commerce websites in terms of efficiency and storage space consumption.

Author's email addresses: H. B. Zhang, haibin.zhang@mq.edu.au, Department of Computing, Macquarie University; Y. Wang, yan.wang@mq.edu.au, Department of Computing, Macquarie University; X. Z. Zhang, xiuzhen.zhang@rmit.edu.au, School of Computer Science and IT, RMIT University; E.-P. LIM, eplim@smu.edu.sg, School of Information System, Singapore Management University.

DOI: http://dx.doi.org/10.1145/0000000.0000000

Categories and Subject Descriptors: H.3.5 [**Online Information Services**]: Commercial services

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: E-Commerce, Trust Management, Trust and Reputation, Contextual Transaction Trust, Online service, Data Warehouse, Aggregation Index

1. INTRODUCTION

In e-commerce environments, when a buyer needs to select a seller from a large pool of sellers, the trustworthiness of a seller is a crucial issue in decision-making [Jøsang et al. 2007; Kim et al. 2008]. At eBay[1] with 233 million sellers and buyers, after each transaction, a buyer can provide a rating (+1, 0, or -1) to the centralized trust management system according to transaction quality. After accumulating over a time period, a single positive feedback rate is calculated to indicate the trustworthiness of the seller in the latest time period (e.g., *"the latest one month"*, *"the latest six months"* and *"the latest twelve months"*). However, this simple trust management system is vulnerable to some frauds from malicious sellers [Kerr and Cohen 2006; Rietjens 2006; Jøsang and Golbeck 2009]. 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. In the literature, this is referred to as the *value imbalance* problem [Dellarocas 2002; Kerr and Cohen 2006; Jøsang and Golbeck 2009], and several real world cases have been reported [Rietjens 2006]. For instance, an Australian deceiver at eBay tricked people for more than AU$10K in total. A Californian deceiver cheated victims in transactions for exceeding US$300K in total.

In view of this problem, Zhang et al. [2011; 2012b] identified the key issues related to *value imbalance* in transactions, as outlined below.

— *The lack of consideration of context in transaction trust evaluation*: 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 [Wang and Lin 2008; Wang and Lim 2008; Li and Wang 2010; Rettinger et al. 2011]. In fact, the *value imbalance* problem is only a type of the *context imbalance* problem [Zhang et al. 2012b] in transactions, where imbalance can also exist in product categories. For example, following a few cases of fraud at Alibaba[2], which supports both B2B and B2C online trading with 50 million users, buyers are explicitly reminded to manually check if the products being offered by a supplier fall into in the same categories as the products that the supplier usually sells[3]. This example also indicates that reputation-based transaction trust evaluation should be "transaction context-aware".

— *The static results of trust evaluation*: Most models compute a single trust value based on past transactions [Sabater and Sierra 2001; Kamvar et al. 2003; Xiong and Liu 2004; Wang and Varadharajan 2005a; Wang et al. 2009; Wang et al. 2012]. However, such a single value basically only reflects a seller's general trust status, and is static with regard to any forthcoming transaction [Wang and Lim 2008]. As illustrated above, different transactions may have different contexts. The static trust evaluation of a seller can hardly predict the likelihood of a successful forthcoming

[1]http://www.ebay.com/
[2]http://www.alibaba.com/
[3]http://resources.alibaba.com/article/232530/Protect_yourself_from_fraudsters_pretending_to_be_Gold_Suppliers.htm

transaction. Thus, trust evaluation should be associated with both past transactions and the forthcoming transaction.

1.1. Motivation

Let us consider a simple example. Suppose a malicious seller S_1 has completed 198 transactions with good quality selling "*AT&T SIM Card*" at the price of \$1 and obtained 198 positive ratings. The seller S_1 also has completed other 2 transactions with poor quality selling "*Apple iPhone5s 16GB*" at the price of about \$700 and obtained 2 negative ratings. Based on the trust evaluation model used at eBay, the trustworthiness of S_1 is as high as 0.99. Next, consider a scenario that a buyer B plans to buy an "*Apple iPhone5s 16GB*". In the meantime, the seller S_1 is selling this product, and the price \$700 offered by him/her is cheaper than other sellers. Clearly, the seller S_1 is very attractive and appears to be trustworthy as well. Thus, buyer B tends to buy the "*Apple iPhone5s 16GB*" from S_1. In such a case, the monetary loss of B is very likely to happen. However, in addition to the general trust value 0.99 to buyers, if B knew that S_1 received negative ratings in occurred transactions selling "*Apple iPhone5s 16GB*", B would not purchase from S_1. In fact, from buyers' point of view, they are more concerned about the trustworthiness of a seller in a potential forthcoming transaction, rather than a general trust value resulting from all past transactions.

Suppose a seller S_2 has completed many more transactions, selling products in a variety of categories over a long period of time. When buyer B plans to buy a "*Canon EOS 6D SLR Digital Camera*" at the price of around \$1600 from S_2, in addition to the trustworthiness of S_2 in selling this product, B could also be concerned about the trustworthiness of S_2 in selling "*Canon DSLR camera*" with a price range of "\$1000-\$2000" (i.e., a query w.r.t. a higher layer in the hierarchical product category in a price range) in *the latest 3 months* or *the latest 6 months*. This is particularly the case when the product in the forthcoming transaction is just available in market and the number of existing transactions selling this product is quite low or even zero. If S_2 is reputable in all these related transactions, there should be good reasons for B to trust S_2 in a new transaction for purchasing a "*Canon EOS 6D SLR Digital Camera*" at the price of around \$1600. Otherwise, if S_2 has problems in the transactions in a certain product category or a certain price range (e.g., S_2 received a lot of negative ratings in the transactions in selling "*Canon EOS 6D SLR Digital Camera*"), it is necessary for trust evaluation to indicate the flaw of S_2 in reputation.

The above process follows the suggestion provided on the Alibaba website, which advises buyers to check if the product to be purchased from a seller is in the categories in which the seller usually sells and if the existing transactions in these categories are reputable. Similarly, as a buyer is very concerned about the possibility of monetary loss, trust evaluation needs to indicate trustworthiness over different price ranges, each of which takes the price of the product to be purchased as approximately the medium value. In addition to them, a further step is to evaluate trust over the combination of product category and price range as well as time period, as different buyers buy products in different categories and with different prices from the same seller. Such evaluation results can reveal potential risk if a seller has problems in reputation in the transactions in a product category, a price range and a time period, related to the potential new transaction that the buyer plans to complete with the seller.

Obviously, these identified needs cannot be satisfied by a single-value trust valuation model. In the meantime, the new needs bring challenges to trust computation as a long-existing seller usually has large-scale transactions with a wide variety of product categories as well as a wide price range. Therefore, the computation of a seller's trustworthiness in various transaction contexts incurs high complexity.

1.2. A Trust Vector based Framework and The Challenges in Computation

Based on the above examples and analysis, in contrast to most existing trust management models that compute a single trust value, our proposed framework is to compute a trust vector for a seller [Zhang et al. 2012b]. The computation of trust values in the trust vector takes transaction context into account and is associated with a forthcoming transaction.

The trust vector consists of three major elements, which are called Contextual Transaction Trust (CTT) values. They are

(1) *the trustworthiness of a seller in selling a specific product to be traded in a forthcoming transaction;*
(2) *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) *the trustworthiness of the seller in a 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 "*Canon EOS 6D SLR Digital Camera*" is the product in the forthcoming transaction, the buyer can specify and adjust *"product category"* along a path in the product category hierarchy, such as *"Canon DSLR camera"*, *"DSLR camera"* and *"Digital camera"* in sequence. If the product is "*Apple iPhone5s 16GB*", the corresponding product categories are "*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 product as approximately the medium value.

We use *granularity* to represent the differences in transaction context determined by a layer in the product category hierarchy, a price range and a time period. In addition, we 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, and thus avoid monetary losses of buyers.

However, at e-commerce websites, a popular seller usually sells a wide variety of products distributed in a number of product categories. In addition, a large number of buyers can be accessing one seller's reputation data simultaneously with regard to their potentially forthcoming transactions. In order to promptly answer a buyer's CTT queries, it is necessary to pre-compute aggregates over large-scale transaction data and ratings with necessary combinations of three context dimensions, i.e., Product Category, Price and Transaction Time. In addition, storing the aggregation results will consume a large volume of space particularly for a system with millions of sellers. Thus, the *CTT computation* for outlining sellers' reputation profiles is a very challenging problem that requires new data structures and novel algorithms that are scalable to large-scale e-commerce websites in terms of efficiency and storage consumption for CTT computation.

1.3. Our Approaches and Contributions

To solve the challenging CTT computation problem, we propose our model *Reputation-Pro* in this paper. Our work and contributions are briefly summarized below:

(1) In contrast to most existing trust evaluation models [Sabater and Sierra 2001; Damiani et al. 2002; Kamvar et al. 2003; Xiong and Liu 2003; Xiong and Liu 2004; Wang and Varadharajan 2005b; Malik and Bouguettaya 2009; Wang et al. 2012], our model considers three important context dimensions in e-commence environments, i.e., *Product Category*, *Price* and *Transaction Time* (see Section 3), and it outlines the reputation profile of a seller which can indicate the dynamic trustworthiness in different products, product categories, price ranges, time periods, and any necessary combination of them (see Section 4).

Generally speaking, *ReputationPro* is typically a heuristic-based [Sherchan et al. 2013] multi-context [Sabater and Sierra 2005] trust evaluation model. There are two important reasons leading to its outperformance over the existing context-aware trust evaluation models:

— *ReputationPro* is a multi-context model which has the mechanisms to deal with several contexts at a time comprising different trust or reputation values associated with them.
  Compared with single-context trust evaluation and similarity-based context-aware trust evaluation, multi-context trust evaluation can reflect a seller's dynamic trustworthiness in various transaction contexts, which provides comprehensive and detailed trust information of a seller. As multi-context trust evaluation is much more complex particularly when considering the combinations of context dimensions, very limited work reported in the literature.
— *ReputationPro* is an efficient heuristic-based multi-context model that can be directly applied in large-scale e-commerce applications.
  Like PeerTrust [Xiong and Liu 2004] and RATEWeb [Malik and Bouguettaya 2009] trust evaluation models, *ReputationPro* adopts heuristic-based technique to aggregate and average trust ratings as the trustworthiness or reputation values of a seller. Compared with IHRTM model [Rettinger et al. 2011], which is the only multi-context model reported in the literature and adopts statistical and machine learning-based techniques, *ReputationPro* is much more efficient and thus more suitable to be applied to the dynamic environments of e-commerce applications with millions of users and transactions that are updated every day.

Note that we have to point out that both rater credibility and secure data storage are important issues in trust evaluation. But they are out of scope of this article. Our proposed *ReputationPro* model has a different focus on how compute reputation profile of sellers efficiently with reduced space consumption.

(2) In the literature, our targeted CTT computation problem is similar to data warehousing and OLAP (On-Line Analytical Processing) technology [Chaudhuri and Dayal 1997] (see Section 2.3). In particular, the traditional RA (Range Aggregate) [Papadias et al. 2001] in two-dimensional spatial data warehouses is relatively close to CTT computation. Typically, an RA query is in regards to the computation of the total number of points falling into a query region. Thus, we first present a review on popular approaches to the RA problem in two dimensional space and identify the limitations of these approaches in solving our targeted problem (see Section 2.4). Then, we further extend the RA problem in a two-dimensional space to CTT computation with the x-axis representing the *Transaction Time* dimension in days, the y-axis representing the *Transaction Amount* dimension, and the *Product Category* dimension taken as the extended third dimension (see Section 5).

(3) Towards efficient CTT computation, we propose a new disk-based index scheme *CMK-tree* and a CTT computation algorithm (see Section 6). According to the re-

quirements of CTT computation, four important and remarkable characteristics of the *CMK-tree* are summarized below:

— In the traditional two-dimensional RA problem [Tao et al. 2004] (see Fig. 2), one point represents one object only (e.g., a car). By contrast, as a common case in e-commerce environments, a seller may have multiple transactions with the same price on a given day selling the same product, i.e., one point may represent multiple such transactions. The *CMK-tree* does not index all transactions but aggregates the repeated transactions on a given day, which sell the same product;
— Some existing approaches to two-dimensional RA problem overlook the inserted objects themselves. Unlike these approaches, the *CMK-tree* guarantees that each specific product can be indexed in order to compute the trustworthiness of the seller in selling a product (i.e., TIST see Section 4.3);
— The CTT computation has the same characteristic as the transaction-time model [Zhang et al. 2008], i.e., the records of newly happened transactions are inserted in chronological order. The *CMK-tree* adopts multi-version structure [Becker et al. 1996] to effectively deal with transaction-time model;
— Only two Vertical Range Aggregate (VRA) queries [Tao et al. 2004] are carried out to answer a CTT query based on the *CMK-tree*. This is more efficient than the *MVSB-tree* which needs to carry out four dominance-sum queries for a RA query.

(4) Though three disk-based approaches taking into account the above special characteristics have been proposed [Zhang et al. 2014] to CTT computation, they have low efficiency of computing CTT values in some cases (see Section 2.6). By contrast, the new index scheme *CMK-tree* proposed in this article reduces computation time by 12.2%-66.7% on four large datasets. In particular, while answering a buyer's CTT queries for each brand-based product category, it has almost linear query performance (see Section 6.4). This is a significant advantage in answering CTT queries when a large number of buyers are accessing a seller's reputation data simultaneously.

(5) Existing approaches for CTT computation adopt a single fine time granularity and aggregate the ratings by days [Zhang et al. 2014]. However, with continuous growth in transaction time (e.g., one year or two years) and significant increase of historical transaction data and ratings, the aggregation index with a single fine time granularity does not scale in terms of storage space. Here the *aggregation index* refers to the index containing some aggregates of ratings. To solve this problem, we further propose an approach *CMK-tree*^{RS} to reduce the storage space allocated to each seller for storing the aggregation index (see Section 7). The *CMK-tree*^{RS} takes into account the requirements of buyers' CTT queries, which maintains the aggregation index at different time granularities: recent ratings (e.g., *"the latest 3 months, i.e., the latest 90 days"*) are aggregated at the fine time granularity of days, and earlier ratings (e.g., *"3 months ago, i.e., 90 days ago"*) are aggregated at a coarse time granularity of weeks.

(6) We have conducted experiments on four large datasets with transactions of 12 months. The experimental results illustrate that the performance of *CMK-tree* is much better in efficiency than all three existing methods [Zhang et al. 2014] in computing CTT values. In addition, our proposed *CMK-tree*^{RS} brings a little loss to the accuracy of CTT computation with much gain in storage space reduction and computation time improvement (see Section 8).

The rest of the paper is organized as follows. Section 2 provides a brief overview of related work. We introduce the modeling of transaction context in Section 3. In Section 4, we introduce our proposed trust vector and the *ReputationPro* model. Section 5 presents how to extend Range Aggregate (RA) in a two-dimensional space to CTT Computation. Section 6 proposes a new disk-based index scheme the *CMK-tree* in detail.

Section 7 proposes an approach *CMK-tree*^{RS} to save storage space of aggregation index. While Section 8 evaluates our approach empirically, Section 9 concludes our paper.

2. RELATED WORK

This section reviews related work in four aspects. First, Section 2.1 presents a review on the existing trust evaluation approaches. In contrast to existing studies [Sabater and Sierra 2005; Jøsang et al. 2007] which introduce typical trust evaluation models, we focus on categorizing trust models from different perspectives. Second, Section 2.2 reviews some existing context-aware trust evaluation approaches. In particular, a table is plotted to compare *ReputationPro* model with some existing trust evaluation approaches so as to highlight its characteristics and the contributions of our work from the perspective of trust evaluation. Third, Section 2.3, Section 2.4 and Section 2.5 review the related techniques in data warehousing. In these sections, we also focus on identifying the limitations of these techniques in resolving our targeted CTT computation problem. Finally, Section 2.6 reviews the existing approaches to CTT computation.

2.1. Taxonomy of Trust Evaluation

2.1.1. Application-Based Taxonomy. In the literature, some works categorize trust evaluation approaches according to their application environments [Wang and Li 2011; Sherchan et al. 2013]. These are generally subdivided into trust evaluation models applied in Network and those applied in Internet. Network applications include Peer-to-Peer (P2P) networks [Suryanarayana and Taylor 2002], multi-agent systems [Sabater and Sierra 2005], social networks [Sherchan et al. 2013] and ad hoc networks [Zhang 2011]. Internet applications include e-commerce [Jøsang et al. 2007], web services [Wang and Vassileva 2007] and cloud computing [Noor et al. 2013].

In P2P networks, Kamvar et al. [2003] propose the EigenTrust model, and a "global" trust value of a given peer is calculated via collecting binary trust ratings. Xiong and Liu [2004] propose a PeerTrust model which defines some general trust metrics and formulas to aggregate ratings into a final trust value. In multi-agent systems, Marsh [1994] proposes a computational model for trust, which is acknowledged as the earliest work about trust in computer science. In Marsh's computational model, the trust properties, such as context dependent and propagative are introduced. In social networks, trust propagation is an important issue. Golbeck and Hendler [2006] propose trust propagation algorithms based on binary ratings. The existing trust models in ad hoc networks focus on modeling the trustworthiness of nodes by collecting trust information about them from other nodes [Liu and Issarny 2004] and delivering reliable packets [Zouridaki et al. 2006]. In the field of service-oriented computing (SOC), Wang et al. [2009] propose some trust evaluation metrics and a formula for trust computation, with which a final trust value is computed. RATEWeb model [Malik and Bouguettaya 2009] aggregates consumers' ratings which aims to facilitate the trust-oriented service provider selection. Based on feedback, Habib et al. [2011] focus on computing the "global" reputation of a cloud service.

2.1.2. Technique-Based Taxonomy. Like the taxonomy proposed in [Damiani et al. 2006; Sherchan et al. 2013], we further attempt to categorize trust evaluation approaches according to the techniques that are adopted for trust establishment.

— *Traditional Security Techniques*: Not surprisingly, trust is related to security, and thus traditional security techniques, e.g., authentication, encryption, access control, etc, can be adopted for trust establishment. For example, Vimercati et al. [2012] proposed an approach to secure data access based on credential-based access control and trust management. Hwang and Li [2010] propose a security-aware cloud architecture, which uses policies to evaluate the credibility of cloud service.

— *Heuristic-Based Techniques*: The trust evaluation models that adopt heuristic-based techniques aim to define a practical model which is easy to understand and construct [Sherchan et al. 2013]. Therefore, they are suitable for systems with a large number of users.
  From the computational point of view, one of the heuristic-based approaches is to aggregate and average quantitative feedback ratings. For example, the models in [X-iong and Liu 2004; Malik and Bouguettaya 2009; Wang et al. 2012] calculate the summation or weighted average of ratings. In addition, the works in [Damiani et al. 2006; Wang et al. 2009] propose new aggregation methods taking advantage of fuzzy models where membership functions are used to determine the trustworthiness of targets.

— *Statistical and Machine Learning-Based Techniques*: The statistical and machine learning-based approaches focus on proposing a reasonable mathematical model for managing or inferring trust information.
  Typically, Bayesian systems [Mui 2003; Jøsang and Ismail 2002] and subjective belief model [Jøsang 2001; Wang and Singh 2007] are two major examples based on s-tatistical theory. On the other hand, machine learning techniques, such as Artificial Neutral Networks (ANNs) [Ham et al. 2009] and Hidden Markov Models (HMM-s) [ElSalamouny et al. 2010], are adopted for trust evaluation. In [Wang et al. 2013], conditional probability model is used to infer the trust values between participants within online social networks.

— *Information Theory-Based Techniques*: In online trading, there is a gap between the committed information, such as product quality, and buyers' actual observations. From the point of view of information theory, Sierra and Debenham [2007] proposed a set of formulas to define commitment and enactment (observation) as well as the concepts like reliability and reputation. Similarly, in social networks, Adali et al. [2010] use entropy to measure "balance in the conversation" between two users, and they define their model as behavior-based trust model.

2.2. Context-Aware Trust Evaluation

2.2.1. The Granularity of Trust Evaluation. As mentioned before, most of trust evaluation approaches lack consideration of context information. Instead, they often compute one value to reflect a general or global trust status of a target [Xiong and Liu 2003; Kamvar et al. 2003; Wang and Varadharajan 2005a; Vu et al. 2005; Wang and Varadharajan 2005b; Wang et al. 2009].

In the literature, some studies differentiate trust evaluation models by granularities, and categorize them into *single-context* models and *multi-context* models [Mui 2003; Sabater and Sierra 2005]. More specifically, the single-context models refer to models that compute a single trust or reputation value, without taking into account the context information. This is the *coarse-granularity* trust evaluation. By contrast, the multi-context models refer to models that have the mechanism to deal with several contexts at a time, while comprising different trust or reputation values associated with them. This is the *fine-granularity* trust evaluation. Compared with single-context models, multi-context trust evaluation models can provide comprehensive and detailed trust information of a target, and so its results are more accurate. But multi-context trust evaluation is much more complex, particularly when considering the combinations of context dimensions. Thus, very limited work has been reported in the literature.

However, one may argue that it is necessary to introduce context information for trust evaluation with high computational complexity. Due to the diversification of a member's performance within certain application environment, the single-context model has its limitations. For example, in e-commerce, as depicted by the motivation at the beginning of this paper (see Section 1.1), *value imbalance* is a typical problem resulting from single-context trust evaluation. Likewise, Liu et al. [2012] identify an "unexpected" phenomenon of reputable sellers in e-commerce called *imprudence*, which refers to the situation where they behave inappropriately (possibly out of complacence to deliver poor products). Therefore, all of these evidences suggest that trust evaluation with fine granularity (multi-context trust evaluation) is in great demand.

2.2.2. Trust Evaluation with Contextual Information. In the literature, there are some existing studies considering the relationship between trust evaluation and context information. In this subsection, we review and categorize them in following three aspects.

— *Multi-faceted Trust Evaluation*: Griffiths [2005] proposes a Multi-Dimensional Trust (MDT) model, which studies contextual trust from a multiple-faceted perspective. The trustworthiness of a particular task can be modeled in several dimensions (e.g., timeliness, quality and cost), letting a user specify the weight of each dimension for trust evaluation based on the personal preference. Thus, given the same seller, the trust results computed for different buyers may vary. Similarly, in REGRET [Sabater and Sierra 2001] and RATEWeb systems [Malik and Bouguettaya 2009], a multi-dimensional structure is adopted when evaluating a seller's reputation.
  However, these models still focus on how to compute a single general or global trust value. From the point of view of granularity, MDT belongs to the single-context model, and it overlooks that the transaction context (e.g., product category and transaction amount) may change in historical transactions. Therefore, they still can hardly predict the likelihood of a successful forthcoming transaction.

— *Similarity-Based Context-Aware Trust Evaluation*: The context similarity calculation is regarded as an important means to deal with the contextual trust evaluation problem. Uddin et al. [2008] propose a CAT (Context-Aware Trust) model to compare the similarity of contexts by using key values that can describe a specific context at least partially. Caballero et al. [2007] defined a formula using task key values to calculate the similarity between two tasks in order to evaluate the trust level of different tasks. Rehak et al. [2006] propose a trust model to resolve contextual trust, using clustering to identify the full context space as several reference contexts based on the context attributes. The trust evaluation of a new context is the weighted sum of the trust values in all reference contexts. Toivonen et al. [2006] use a more complex ontology structure to calculate similarity.
  Similarity-based context-aware trust evaluation models still do not belong to multi-context trust evaluation. As pointed out by Mui [2003], the context similarity is used to infer the trustworthiness of a target in a certain context where there are no or not enough ratings from the same context. Therefore, these trust models still focus on calculating a single trust value under corresponding specific context. However, they are different from our *ReputationPro* model proposed in this article, which aims to promptly answer a buyer's CTT queries on a seller's trustworthiness in various transaction contexts. Instead of providing only a single value, our approach computes sets of trust values to outline the reputation profile of a seller.

— *Statistical and Machine Learning-Based Context-Aware Trust Evaluation*: Rettinger et al. [2011] propose a context-sensitive trust evaluation model (IHRTM) taking advantage of statistical relational learning. In the IHRTM model, contextual information is discussed in the $Seller \times Item$ space. According to the learning results, all 47

selected sellers in their experiments are assigned to 4 groups based on the context attributes in the $Seller$ space, such as feedback score and positive feedback rate, and the 630 items sold by these sellers are assigned to 40 clusters based on the context attributes in the $Item$ space, such as product category and product condition (new or used). Finally, a 4×40 matrix is formed to indicate the trustworthiness of 4 seller clusters under 40 item clusters. Note that the IHRTM model belongs to multi-context trust evaluation. In order to improve the accuracy of predicting the trustworthiness of a forthcoming transaction, Liu and Datta [2012] extract useful features from transaction context, such as product category and price, as observations to construct a Hidden Markov Model (HMM) for modeling the dynamic trust of a seller. In addition, to reduce computational complexity, information theories and Multiple Discriminant Analysis (MDA) are adopted for feature space reduction.
A major disadvantage of all the statistical and machine learning-based trust evaluation approaches is their high computational complexity, which makes them difficult to be applied to the environments with millions of users [Sherchan et al. 2013]. For example, when having a large number of sellers, there will be many clusters of sellers for IHRTM, leading to a high complexity in learning iterations. For dynamic environments of e-commerce applications where new transactions happen every day, the cost of re-learning, which takes new transactions into account, is higher. Moreover, from the transaction context perspective, IHRTM does not support the analysis of reputation on the product categories along a path in the product category hierarchy (e.g., "*Apple iPhone*" and "*Smartphone*" in sequence). This analysis is particularly necessary when a new product, or a product in a new category, is just released. Also, it does not support the trust evaluation of a seller in the transactions in a given price range. This need comes from a buyer when s/he is very concerned about the risk of monetary loss in a forthcoming transaction [Xiong and Liu 2004; Swaminathan et al. 2010]. By contrast, the *ReputationPro* model can outline a seller's reputation profile and indicate his/her dynamic trustworthiness in different product, product categories, price ranges, and time periods.

In Fig. 1, we draw a table to compare *ReputationPro* trust evaluation model with some existing trust evaluation approaches so as to highlight its characteristics and the contributions of our work from the perspective of trust evaluation.

2.3. OLAP (On-Line Analytical Processing) and Data warehouses

In the broader research literature, our targeted problem of CTT computation is somewhat similar to sales analysis from multiple perspectives in data warehouses and business intelligence. Typically, the sales data warehouse for a company contains three dimensions Product category, Location and Time. The OLAP operations refer to the queries on the aggregation of sales over each dimension or their combinations, such as the sum of sales per product category or the sum of sales per product category and per month combination. Gray et al. [1996] point out that there are $O(2^n)$ possible aggregations for a data warehouse with n dimensions composing a "data cube".

In order to accelerate query processing, some results can be pre-computed and stored as *materialized views* [Harinarayan et al. 1996; Lin and Kuo 2004]. However, these approaches only benefit the queries on dimensions with predefined hierarchies. In particular, the Time dimension in sales analysis refers to static calender months, e.g., January, February, etc. By contrast, in CTT computation, while the product category hierarchy is predefined and static, Price and Transaction Time are dynamic dimensions (see Section 3). Specifically, the dynamicity of the Price dimension refers to the reality that the price of a product may change over time. Even on a given day, multiple transactions selling the same product may have different prices. In addition, the

Approaches	Application Fields	Technology Adoption	Contextual Information
Marsh [1994]	MAS/SN	HET	CSI
EigenTrust- Kamvar et al. [2003]	P2P	HET	SC
Jøsang [2001]	MAS/SN	ST/ML	SC
PeerTrust- Xiong and Liu [2004]	P2P, EC	HET	CSI
Vimercati et al. [2012]	WS	TST	SC
RATEWeb- Malik and Bouguettaya [2009]	WS, EC	HET	SC
Golbeck and Hendler [2006]	MAS/SN	HET	SC
Wang et al. [2013]	MAS/SN	ST/ML	CSI
Liu and Issarny [2004]	AHN	HET	CSI
Hwang and Li [2010]	CC	TST	SC
TRSIM- Caballero et al. [2007]	P2P	HET	CSI
MDT- Griffiths [2005]	MAS/SN	HET	SC
Liu and Datta et al. [2012]	EC	ST/ML	CSI
REGRET- Sabater and Sierra [2001]	MAS/SN	HET	SC
Sierra and Debenham [2007]	MAS/SN, EC	IT	CSI
Wang and Singh [2007]	MAS/SN	ST/ML	SC
Wang et al. [2009]	WS, EC	HET	SC
IHRTM- Rettinger et al. [2011]	EC	ST/ML	MC
RepuationPro	EC	HET	MC

Application Fields	Technology Adoption	Contextual Information
P2P: **P**eer to **P**eer networks MAS/SN: **M**ulti-**A**gent **S**ystems and **S**ocial **N**etworks AHN: **A**d-**h**oc **N**etworks EC: **E**-**C**ommerce WS: **W**eb **S**ervices CC: **C**loud **C**omputing	TST: **T**raditional **S**ecurity **T**echniques HET: **He**uristic-Based **T**echniques ST/ML: **St**atistical and **M**achine **L**earning-based **T**echniques IT: **I**nformation **T**heory-Based Techniques	SC: **S**ingle-**C**ontext model (without taking into account context Information or multi-faceted trust evaluation) CSI: **C**ontext information or context **S**imilarity as **I**mpact factor (no in-depth discussions on the impact of context and still not a multi-context trust evaluation) MC: **M**ulti-**C**ontext model (computing several contexts at a time and comprising different trust or reputation values associated with them)

Fig. 1. The comparison of existing trust evaluation approaches

dynamicity of the Transaction Time dimension refers to the new transactions added to the database over time, modifying the set of "most recent transactions".

Some existing works improve the performance of queries in data based on specifically designed column-oriented database systems [Abadi et al. 2008]. Different from these works, our approach in this article is based on popular relational database management systems that are being widely used by e-commerce websites, so that the designed models can be directly applied.

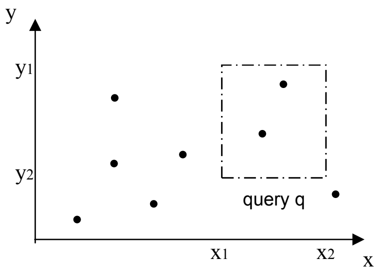


Fig. 2. An example of an RA query

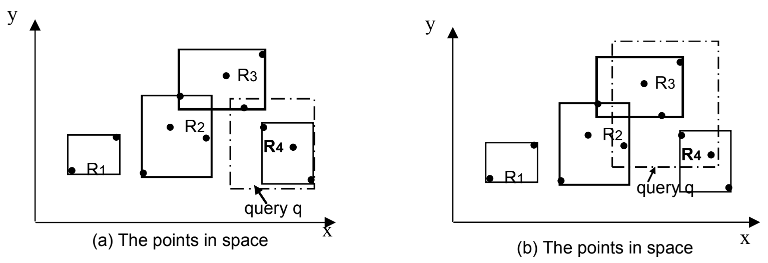


Fig. 3. An aR-tree

2.4. The Two-Dimensional (2D) RA Query

In the literature, the RA (Range Aggregate) problem in two-dimensional spatial data warehouses is relatively close to our targeted CTT computation problem (see Section 5); therefore we review the approaches to the RA problem separately.

Fig. 2 shows the traditional RA query [Papadias et al. 2001] in a two-dimensional space which is in regards to computing the total number of points falling into a query region q surrounded by $[x_1, x_2]$ and $[y_1, y_2]$, e.g., answering a query in traffic supervision systems, such as: "What is the total number of cars inside a certain district?". Usually, a query region can be any area within the two-dimensional space. To accelerate query processing, most existing works still pre-compute some results, but they appropriately store the results in the specialized index [Papadias et al. 2002; Tao et al. 2004; Tao and Papadias 2005; Zhang et al. 2008]. Since the memory-based approaches are inappropriate for large-scale data processing, in this subsection, we will restrict our review to some well-known disk-based approaches.

2.4.1. The aR-tree. The *aR-tree* [Jurgens and Lenz 1998; Papadias et al. 2001] maintains the x-y coordinates for each minimum bounding rectangle (MBR) (e.g., R_1, R_2, R_3, R_4 in Fig. 3(a) are all MBRs). In the meantime, each MBR records the total number as an aggregate of the objects that fall into an MBR. To compute the number of objects in a query region q, in Fig. 3(a), the MBR R_4 within q will not be accessed. Rather, R_4's pre-computed aggregate (i.e., 3) is directly used. But R_3 needs to be visited, as it partially overlaps with q. The total number of points in q equals their sum $3 + 1 = 4$. A serious problem of the *aR-tree* is that its performance significantly degrades when answering a large query region, since in such a case there are more MBRs overlap with the query region (see Fig. 3(b)).

2.4.2. The aP-tree. Tao et al. [2004] propose the *aP-tree* to improve the *aR-tree*, based on the following transformation on a query region q. They first convert each spatial point to an *interval* (i.e., a horizontal line) (see Fig. 4(b)). When q is surrounded by $[x_1, x_2]$ and $[y_1, y_2]$ is transformed to two *borders* (i.e., two vertical lines): $x_1 : [y_1, y_2]$

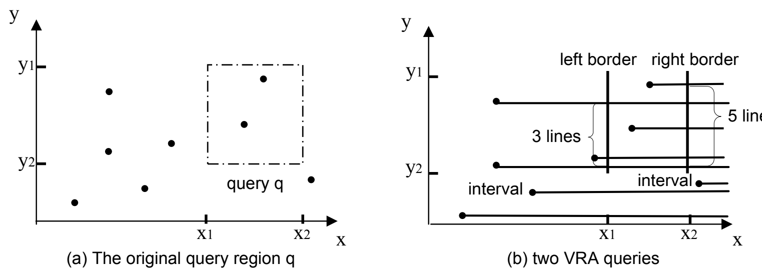


Fig. 4. An RA query transformed to two Vertical Range Aggregate (VRA) queries

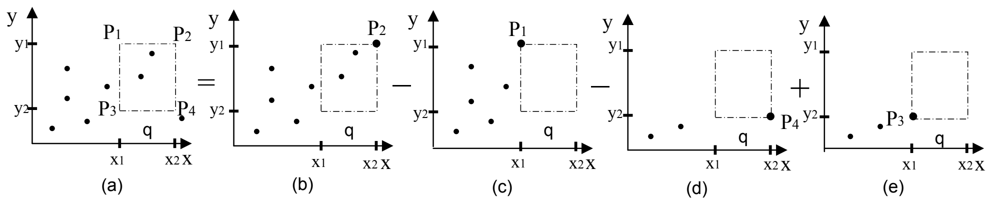
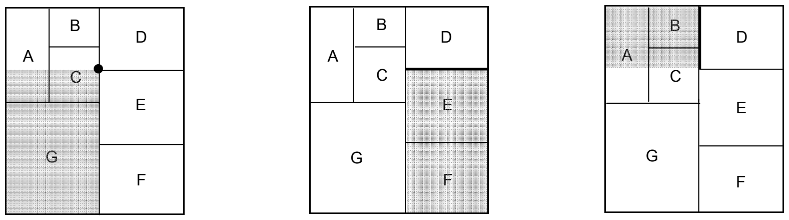


Fig. 5. An RA query transformed to four dominance-sum queries

and $x_2 : [y_1, y_2]$, an RA query is converted to retrieving the number of intervals that intersects the two borders. For instance, in Fig. 4, the number of intervals intersecting the left border $x_1 : [y_1, y_2]$ is 3 while the number of intervals intersecting the right border $x_2 : [y_1, y_2]$ is 5. The total number of points in q equals their difference $5-3 = 2$. Tao et al. [2004] define the number of intervals intersecting a border as a Vertical Range Aggregate (VRA). In order to compute each VRA value, the *aP-tree* is then proposed that contains an additional field *agg* in each entry, extending the original *multiversion B-tree (MVBT)* [Becker et al. 1996].

In order to answer an RA query, the *aP-tree* indexes all the objects [Zhang et al. 2008; Zhang et al. 2014]. As shown in Fig. 2, in the traditional RA problem, one point in a two-dimensional space represents only one object (e.g., a car). However, in our targeted CTT computation problem, one point may represent multiple such objects. For example, it is quite common that a seller has multiple transactions with the same price selling the same product on a day. Here the x-axis represents days, and the transactions occurred on the same day have the same x-coordinate. In this case, the proposed new index scheme should take this characteristic into account. Unlike the *aP-tree*, the new index does not need to index all the transactions; rather, multiple repeated transactions should be aggregated, and then the new index only needs to store the aggregation results.

2.4.3. The MVSB-tree and The BA-tree. Zhang et al. [2008] address the RA problem in a two-dimensional space by converting an RA query to four dominance-sum queries. Given two two-dimensional points $x = (x_1, x_2)$ and $y = (y_1, y_2)$, x dominates y if $x_1 \geq y_1$ and $x_2 \geq y_2$. The corresponding dominance-sum of the point P is the aggregation of all the points that are dominated by P. Therefore, in Fig. 5(a), the total number of points in the query region $P_1P_2P_3P_4$ equals $7-5-2+2 = 2$, namely, the dominance-sum of the point P_2 (see Fig. 5(b)) subtracts the dominance-sum of the point P_1 (see Fig. 5(c)) and the dominance-sum of the point P_4 (see Fig. 5(d)). As the dominance-sum of the point P_3 (see Fig. 5(e)) has been subtracted twice, their sum must be added again. In order to compute each dominance-sum query, Zhang et al. further propose the *MVSB-tree* [2001; 2008] and the *BA-tree* [2002], respectively.



(a) the subtotal of points to the lower left of D

(b) the X-coordinates of the points below D

(c) the Y-coordinates of the points to the left of D

Fig. 6. The structure of *BA-tree*

The *MVSB-tree* results from augmenting the *SB-tree* [Yang and Widom 2003]. It logically divides the two-dimensional space into multiple nonintersecting rectangles. When inserting an object with the coordinate (x_i, y_i), the aggregation operations perform in all the rectangles within the area $[x_i, max_x) \times [y_i, max_y)$. Here max_x and max_y collectively form the upper-right corner of the complete space. Although the *MVSB-tree* considers a point may represent multiple such objects, it overlooks the inserted objects themselves. If it is applied to CTT computation problem, each specific product cannot be indexed. As a result, computing the trustworthiness of a seller in selling a product cannot be fulfilled. Furthermore, the *MVSB-tree* is particularly designed for solving the dominance-sum problem. Thus, four dominance-sum queries are needed to answer an RA query (see Fig. 5). By contrast, our proposed new index scheme answers only two VRA queries for the same purpose, and thus improves efficiency (see Fig. 4).

The *BA-tree* is another index scheme for answering RA queries that extends the *K-D-B-tree* [Robinson 1981]. Fig. 6 depicts a general structure of *BA-tree*, as in the *K-D-B-tree*, each node corresponds to a rectangular space, such as the area A. The node at a higher level corresponds to a larger rectangular space formed by several adjacent areas, such as the area formed by A, B and C. The root node corresponds to the complete space. The augmentation of the *BA-tree* over the *K-D-B-tree* is that each node (e.g., the area D) also stores three aspects of information: the subtotal of points to the lower left of D (see Fig. 6(a)), the x-coordinates of the points below D (see Fig. 6(b)) and the y-coordinates of the points to the left of D (see Fig. 6(c)). The *BA-tree* achieves linear performance when answering each dominance-sum query. However, as pointed out by Zhang et al. [2008], it does not fit the transaction-time model where the records of transactions are inserted in chronological order.

2.4.4. The CRB-tree. Apart from the above reviewed approaches, the *CRB-tree* has been proposed for solving RA problem [Govindarajan et al. 2003; Agarwala et al. 2012]. The general structure of a *CRB-tree* contains two parts: 1) one normal B^+*-tree* [Bayer and McCreight 1972] constructed on the y-coordinates of points in a two-dimensional space; and 2) another B^+*-tree* constructed on the x-coordinates of points, but with each internal node storing weights as a secondary structure. The *CRB-tree* has good performance for answering RA queries and can further reduce the space consumption. However, as pointed out in [Tao et al. 2004; Zhang et al. 2008], it is based on a stringent assumption that it runs on top of *bit-wise machines*. Specifically, an integer with a value v is represented by exactly $\log_2 v$ bits for a bit-wise machine so that multiple integers may be compressed into a single word. By contrast, a typical *word-wise machine* model uses four bytes to store a single integer. Therefore, the *CRB-tree* "is mainly of theoretical interest" [Tao et al. 2004; Zhang et al. 2008], and does not apply to the

Table I. The summary of limitations

Approaches	Limitations
aR-tree	Performance degrades with a larger query region
aP-tree	Cannot essentially aggregate the same objects
MVSB-tree	Overlooks the inserted objects themselves Based on four dominance-sum queries
BA-tree	Does not fit the transaction-time model
CRB-tree	Does not fit the common word-wise machines

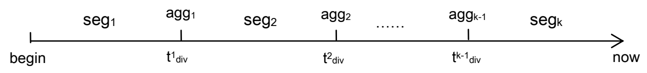


Fig. 7. A general structure of HTA^{FS} model

prevalent commercial word-wise computers. By contrast, our proposed approaches in this article are all based on common word-wise machine models and thus can directly apply to commercial web applications like eBay on tops of commercial servers.

Table I summarises the limitations of existing approaches to two-dimensional (2D) Range Aggregate (RA) after being extended to solve CTT computation problem.

2.5. Hierarchical Temporal Aggregation with Fixed Storage Space (HTA^{FS})

Broadly speaking, the two-dimensional RA problem includes range-temporal aggregation, i.e., point aggregate in a two-dimensional space with one as the time dimension. In addition, a more general problem over RA is spatio-temporal aggregation, i.e., three dimensions with one as the time dimension. In the literature, some index schemes have been proposed to solve these aggregation problems [Zhang et al. 2001; Zhang et al. 2002; Tao and Papadias 2005]. However, all these approaches do not have restriction on storage space to store aggregation index. As a result, with continuous growth in the Time dimension (e.g., one year or two years) and significant increase of historical data, the aggregation index does not scale in terms of storage space.

In contrast to the above existing approaches, Zhang et al. [2003] propose a Hierarchical Temporal Aggregation model with fixed storage space (denoted as *HTA*^{FS}) to control storage space of aggregation index over data streams. Fig. 7 depicts the general structure of the *HTA*^{FS} model to deal with point aggregation in a one-dimensional space. A k-level time hierarchy, where $gran_1$ is at the coarsest time granularity (e.g., by days) and $gran_k$ is at the finest granularity (e.g., by seconds). Suppose that the *HTA*^{FS} model divides the time space [*begin*, now) into k segments. Each segment seg_i ($i = 1, 2, ..., k$) maintains the corresponding aggregations with the time granularity $gran_i$. The term *begin* denotes the starting time and *now* denotes the increasing current time. New objects are inserted with the point "*now*" moving to the right in the x-axis. The constraint for the *HTA*^{FS} model is that the size of available storage space is fixed. When the size of the total storage becomes more than a threshold S, older information is aggregated at a coarser time granularity.

In [Zhang and Wang 2013], we have applied the *HTA*^{FS} model to CTT computation and proposed a model named *CTT*^{FS}. Compared with original *HTA*^{FS} model, the additional characteristics in CTT computation are identified. Specifically, as a seller has imbalanced transaction volumes in different product categories, a dynamic storage space allocation strategy is first proposed in order to guarantee a fixed storage space allocated to a seller for CTT computation. However, although the *CTT*^{FS} model can save the storage space to some extent, as analyzed in Section 7, the strategy of allocating the fixed storage space is unreasonable for CTT computation. Thus, unlike the CTT^{FS} model, a new solution *CMK-tree*^{RS} is proposed to reduce storage space consumption in this article which aggregates ratings with different time granularities for different time periods.

2.6. Existing Approaches to Contextual Transaction Trust Computation

In our previous work, we have proposed a preliminary trust vector which takes transaction context factors into account in the computation of contextual transaction trust values, and have introduced the first set of technical solutions to compute CTT values [Zhang et al. 2012a]. In [Zhang et al. 2012b], more details of our proposed trust vector based framework for CTT computation are presented. In particular, we have conducted empirical studies to compare the trust vector with some typical single-value trust valuation models [Sabater and Sierra 2001; Wang and Varadharajan 2005b] to illustrate its advantages. After that, we have further extended our work [Zhang et al. 2012a] by introducing a new product category hierarchy for supporting finer-grained analysis on the transaction trust of a seller as well as aggregating repeated transactions and have proposed three new disk-based index schemes *eaR-tree*, *eaP-tree* and *eH-tree* for CTT computation [Zhang et al. 2014]. All these approaches can meet the requirements of answering a buyer's CTT queries on the dynamic trustworthiness of a seller in different product categories, price ranges and time periods. However, they have poor performance in some cases. Specifically, as the *eaR-tree* extends the *aR-tree*, the query cost for *eaR-tree* depends on the size of the CTT query region formed by the price range and transaction time range: the larger the query region, the worse the performance in answering a CTT query. For both the *eaP-tree* and the *eH-tree* that extend the *aP-tree*, they index all the transactions and cannot essentially aggregate repeated transactions that occurred on a day, leading to inferior performance.

In the new disk-based index scheme *CMK-tree*, the above problems will be solved. Like the *BA-tree* [Zhang et al. 2002], the *CMK-tree* extends the two-dimensional *K-D-B-tree* or *2-D-B-tree*, however, it adopts a different extension strategy that is particularly designed to efficiently support CTT computation. Moreover, all existing approaches aggregate the transaction data and ratings at the granularity of days. As mentioned in Section 2.5, they lead to the problem of large space consumption when dealing with a large number of sellers. Therefore, we further propose the *CMK-tree*^{RS} to reduce the storage space allocated to each seller for storing the aggregation index.

3. TRANSACTION CONTEXT

This section first presents transaction context dimensions for evaluating the trustworthiness of sellers. Then, we further explain the transaction context imbalance problem existing in e-commerce environments.

3.1. Transaction Context Dimensions

In our previous work [Zhang et al. 2012b], we have identified three important *context dimensions* with influence on the trustworthiness of a forthcoming transaction. They are *Product Category*, *Transaction Amount* and *Transaction Time*. 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.

• **Product Category (a static but hierarchical dimension)**: The category of transaction items has a hierarchical structure. There are some *Products and Services Categorization Standards (PSCS)* that aim at constructing the product category hier-

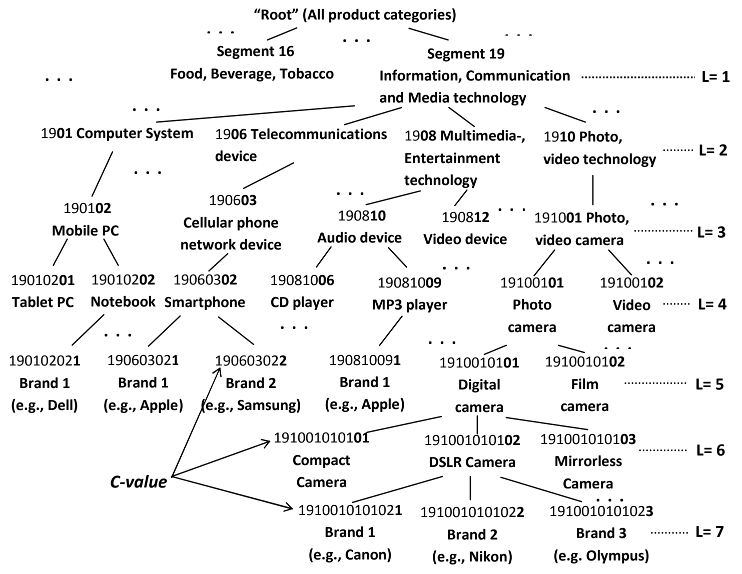

"Root" (All product categories)
Segment 16 Food, Beverage, Tobacco
Segment 19 Information, Communication and Media technology
L= 1
1901 Computer System
1906 Telecommunications device
1908 Multimedia-, Entertainment technology
1910 Photo, video technology
L= 2
190102 Mobile PC
190603 Cellular phone network device
190810 Audio device
190812 Video device
191001 Photo, video camera
L= 3
19010201 Tablet PC
19010202 Notebook
19060302 Smartphone
19081006 CD player
19081009 MP3 player
19100101 Photo camera
19100102 Video camera
L= 4
190102021 Brand 1 (e.g., Dell)
190603021 Brand 1 (e.g., Apple)
190603022 Brand 2 (e.g., Samsung)
190810091 Brand 1 (e.g., Apple)
1910010101 Digital camera
1910010102 Film camera
L= 5
191001010101 Compact Camera
191001010102 DSLR Camera
191001010103 Mirrorless Camera
L= 6
C-value
1910010101021 Brand 1 (e.g., Canon)
1910010101022 Brand 2 (e.g., Nikon)
1910010101023 Brand 3 (e.g. Olympus)
L= 7

Fig. 8. Part of product category hierarchy for the segment *"Information, communication and media"*

archy, such as UNSPSC[4] and eCl@ss[5], each of which groups similar products and provide an industry-neutral hierarchical structure of product categories with up to four layers. eBay has a different product category[6] schema with simply two layers, and it groups products by considering some factors such as marketing and common use.

We establish the product category hierarchy for the analysis of dynamic reputation of a seller. We extends eCl@ss due to its reasonable classification in practice, i.e., products in eCl@ss are more functionally grouped, and they are subdivided for specific usage. For example, in eCl@ss, "*Digital Camera*" can be further classified as "*DSLR (Digital single-lens reflex cameras)*", "*Compact Digital Camera*" and "*Mirrorless Digital Camera*". But it is not subdivided in both UNSPSC and eBay. In addition, we sort out the logical relations between product categories in eCl@ss. Then, we add the attribute "*Brand*" to the product category hierarchy to support finer-grained analyses on transaction trust with "*drill down*" and "*roll up*" operations in the hierarchy. Under each brand, there are corresponding products that belong to this brand.

Fig. 8 presents a small part of our extended product category hierarchy. For instance, if the product is "*Apple iPod nano 16GB (mc696ll/a)*", then its ancestors in the product category hierarchy are "*Apple MP3 player (iPod)*" and "*MP3 player*" in sequence. If the product is "*Apple iPhone5 16GB*", then its ancestors in the product category hierarchy are "*Apple iPhone*" and "*Smartphone*" in sequence. The category hierarchy for the product "*Canon EOS 6D SLR Digital Camera*" is complex that has sever layers, and its ancestors are "*Canon DSLR camera*", "*DSLR camera*" and "*Digital camera*" in sequence. In our extend hierarchy, each product category has a unique id termed as *C-value* (see Fig. 8), with which a layer, the layer's parent and children can be located.

• **Transaction Amount and Transaction Time (two dynamic linear dimensions)**: Transaction amount refers to the sum of the prices of all products in a transaction. A transaction of about $10 is obviously different from one involving $10K. The

[4]http://www.unspsc.org/
[5]http://www.eclass.de/
[6]http://pages.ebay.com/sellerinformation/ebaycatalog/categories.html

larger the transaction amount, the more likely a fraudulent action may occur since the potential benefit of the fraud is greater [Ba and Pavlou 2002]. For the sake of simplicity, like eBay, *each item in a transaction is considered separately* in our work. A transaction with multiple transaction items are taken as several transactions separately with one item each. Hence, the transaction amount equals the price of the product in a transaction. In this paper, we use "transaction amount" and "price" interchangeably. Transaction time is the time when a transaction happens. Trust evaluation is time-sensitive, because the transaction quality may change over time [Spitz and Tuchelmann 2009].

The Price dimension is dynamic as the price of a product may vary from time to time. Owing to product condition (new and used) and product value changes over time, the prices of transactions selling the same product may be different. Also, a buyer's queries on a price range can vary with the same product, or vary from product to product. For instance, the price of a product that a buyer wants to buy from a seller is around $\$500$, and the buyer may be concerned about the trustworthiness of this seller on selling products at a price range of "$400-$600" or "$350-$650". If the price of another product is $\$1500$, the corresponding price range in a query may be "$1000-$2000", taking $\$1500$ as the medium value.

The Transaction Time dimension has a specific characteristic in trust computation. Any CTT queries on the transaction time range starts from a previous point and ends at the point "*now*", i.e., a query regarding the reputation in the recent transactions, such as *"the latest 1 month"*, *"the latest 3 months"* and *"the latest 12 months"*. In addition, Transaction Time dimension is also dynamic because the time point "*now*" changes everyday, and new transactions added to the database over time change the set of "most recent transactions".

3.2. Transaction Context Imbalance

Malicious sellers and fraudulent transactions could take advantage of the results delivered by transaction trust evaluation without considering any transaction context. Consequently, it may lead to some transaction context imbalance problems.

• **Transaction Amount Imbalance**: There are two different cases in the transaction amount imbalance.

(a) A seller accumulates a high level of trust by offering cheap and attractive products, and then s/he may deceive buyers with expensive products. In the literature, this issue is also termed as value imbalance [Dellarocas 2002; Kerr and Cohen 2006; Jøsang and Golbeck 2009].

(b) Buyers usually believe that if a seller has successfully finished many transactions selling expensive products, s/he may not cheat in forthcoming transactions selling cheaper products. In fact, such a "reputable" seller may not be as prudent as in expensive transactions to serve each buyer well due to limited profit.

• **Product Category Imbalance**: A seller has accumulated a high trust level by selling certain products, and then s/he can utilise this high trust value to sell products in different categories for more profit. According to the suggestion from Alibaba (see Section 1.1), such a seller should have different trust levels with respect to different products or different product categories [Xiong and Liu 2004]. For instance, a seller, who sold *watches* before and now starts to sell a certain type of *notebook computers*, should not have the same level of trust as before due to the lack of sufficient experience and reputation in selling the new products with a completely different nature.

4. A TRUST VECTOR BASED FRAMEWORK FOR OUTLINING REPUTATION PROFILE

In this section, we first define the data that are needed for CTT computation. We then present a trust vector which consists of three CTT values and introduce why they can be used to outline the reputation profile of a seller.

4.1. Trust Data Representation

The following data elements are needed for CTT computation.

$$TR^{(t)} =< S; B; p; \textit{C-hrchy}; ta; t; r > \tag{1}$$

- $TR^{(t)}$ is a transaction between a seller S and a buyer B happening at time t;
- p is the product (i.e., transaction item) traded in the transaction $TR^{(t)}$;
- *C-hrchy* represents the path in product category hierarchy to which p belongs;
- ta is the transaction amount in transaction $TR^{(t)}$ for p;
- r is a rating (an integer in a range, e.g., $\{-1, 0, 1\}$ or $\{1, 2, 3, 4, 5\}$) that the buyer B gives to the seller S for $TR^{(t)}$ to reflect a seller's performance during the whole transaction;
- A set of n past transactions can be denoted as $Trans = \{TR^{(t_1)}, TR^{(t_2)}, ..., TR^{(t_n)}\}$.

4.2. CTT metrics

ReputationPro can be directly applied in large-scale e-commerce applications. Thus, like trust evaluation models [Sabater and Sierra 2001; Kamvar et al. 2003; Xiong and Liu 2003; Xiong and Liu 2004; Wang and Varadharajan 2005a; Wang and Li 2011; Malik and Bouguettaya 2009], *ReputationPro* adopts heuristic-based technique to aggregate and average trust ratings as the trustworthiness or reputation values of a seller. Namely, we calculate each CTT value as the average of the ratings in a specific transaction context. Following this idea, two aggregates are pre-computed 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 within 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 $T = \frac{\sum sum\_r_i}{\sum count\_r_i}$.

4.3. A Trust Vector and ReputationPro

Our proposed trust vector [Zhang et al. 2012b] consists of three major CTT values.

(a) Transaction Item Specific Trust (TIST): TIST is the average of all the ratings $\{r_i\}$ in the past transactions $Trans$ for trading the same transaction item p as in a forthcoming transaction.

(b) Product Category based Trust (PCT): PCT is the average of all the ratings $\{r_i\}$ of the past transactions $Trans$ for selling the products in a product category (e.g., "*Canon DSLR cameras*" or "*DSLR cameras*") of p (e.g., "*Canon 6D DSLR camera*") in the product category hierarchy (see Fig. 8). When computing PCT, a price range covering the price ta and a time range can be specified as the parameters. The variables p and ta come from the context of the forthcoming transaction.

(c) Similar Transaction Amount based Trust (STAT): STAT is the trust value of a seller in a specific price range covering price ta and a time range. STAT is important for analyzing the trustworthiness of a seller in different price ranges.

In the above trust vector, all three CTT values are associated with both past transactions and the forthcoming transaction. With the same seller but a different forthcoming transaction, the computed trust values may be different. Even with the same forthcoming transaction, the trust values can vary. This is because a buyer can specify and change the layer in the product category hierarchy, the price range and the time period for computing the last two CTT values: PCT and STAT. With the combinations of the parameters specified in all three context dimensions, different sets of CTT values can be computed, all of which can outline the reputation profile of the seller indicating the trustworthiness in various types of transactions. Thus, our *ReputationPro* model can greatly help detect the possibility of context imbalance problem in a forthcoming transaction.

5. EXTENDING TWO-DIMENSIONAL RANGE AGGREGATE FOR CTT COMPUTATION

In this section, we first discuss the relationship between the two-dimensional RA problem and our targeted CTT computation problem. In Section 3, we have introduced that transaction context includes a static and hierarchical dimension, i.e., Product Category, and two dynamic linear dimensions, i.e., Transaction Amount (Price) and Transaction Time. When computing PCT and STAT values, a CTT query covers both the Transaction Amount dimension and the Transaction Time dimension. Similar to the case depicted in Fig. 2 in Section 2.4, a CTT query can be first 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. 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 $[ta_1, ta_2]$.

Here, for further analysis, we need to point out that each point in a two-dimensional space represents one transaction or a set of transactions. There are three cases that should be differentiated (see Fig. 2).

Case 1: *Given one point at (t_i, ta_i), it may represent only one transaction that occurred on a day t_i with the transaction amount ta_i.*

Case 2: *As mentioned in Section 1.3, 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. In such a case, we need data structures to aggregate these repeated transactions.*

Case 3: *Given one point at (t_i, ta_i), it may represent a set of transactions that occurred on a given day t_i selling different products with the same price ta_i. In such a case, they should be regarded as different transactions and aggregated separately.*

Note that if the prices of transactions selling the same product are different, we regard them as different transactions and aggregate them separately.

We discuss next how to extend the two-dimensional RA problem to CTT computation after taking into account the Product Category as the third dimension:

Step 1: Each transaction has a numeric string *C-hrchy* to represent the path in the product category hierarchy to which the product traded in the transaction belongs;

Following the eCl@ss introduced in Section 3, a two-digit number is added to each layer of the product category hierarchy. Thus a unique *C-value* is assigned to each product category. For example, in Fig. 8, the node "*MP3 player*" at layer 4 has the *C-value* of "19081009" representing the path from the "product category root" to it. As the products traded in the transactions are at the bottom of product category hierarchy, the value of *C-hrchy* equals the *C-value* in the corresponding brand-based product category;

Step 2: Each product category in the product category hierarchy maintains the aggregates $count\_r$ and sum_r that are obtained from the past transactions selling the products in this product category as well as the corresponding transaction amount range and transaction time range;

Step 3: Each product category is an intermediate node in the product category hierarchy. In the meantime, for each brand-based product category (e.g., "*Canon SLR Digital Camera*"), it is the root of a subtree that is external to the hierarchy.

This subtree can be regarded as a tree for solving the RA problem in a two-dimensional space, which records the pairs of $count\_r$ and sum_r in the Transaction Amount dimension and the Transaction Time dimension. Accordingly, as we take the points depicted in Fig. 2 as transactions, all these transactions should belong to the same brand-based product category. Also, the subtree can be of multiple layers, depending on the number of transactions and the distributions in transaction amount and transaction time in the corresponding brand-based product category.

6. THE CMK-TREE

This section presents the *CMK-tree* — a disk-based index structure that supports efficient computation for a buyer's CTT queries. While Section 6.1 describes the structure of the *CMK-tree*, the process of *CMK-tree* construction is provided in Section 6.2. Section 6.3 introduces the retrieval process in a *CMK-tree* to answer a typical CTT query. Here a typical CTT query refers to computing the value of *Product Category based Trust* (PCT) covering three transaction dimensions, namely, the average of all the ratings in the past transactions selling the products at a specific layer in the product category hierarchy, within a price range and a time period (*see details in Section 4.3*). In fact, the process of computing Transaction Item Specific Trust (TIST) or Similar Transaction Amount based Trust (STAT) is very similar to that of PCT. The only difference is that in the computation of TIST, there is a need to further search the actual records in the database. In the computation of STAT, search needs to be performed in all the categories of the products sold by the seller. Section 6.4 focuses on detailed analysis on the structure of the *CMK-tree* and the performance of the *CMK-tree* algorithm in answering buyers' CTT queries. Finally, Section 6.5 discusses how to extend our proposed *CMK-tree* for CTT computation.

6.1. The Structure of the CMK-tree

There are three types of nodes in the *CMK-tree*: *R-node* (Rn), *I-node* (In) and *L-node* (Ln), each of which can have multiple records depending on the node capacity. Generally speaking, as shown in Fig. 9, 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-nodes, and an *MK-tree* consists of I-nodes and L-nodes. Next, we will present the node structures in detail.

6.1.1. The C-tree (Product Category Tree). Following Step 2 in the extension process described in Section 5, each record in an R-node Rn_i (see Fig. 10(a)) has the form

$$< \textit{C-value}, [ta_{min}, ta_{max}], [t_{min}, t_{max}], count\_r, sum\_r, pointer >,$$

where the *C-value* denotes the unique id of the product category within the product category hierarchy; $[ta_{min}, ta_{max}]$ and $[t_{min}, t_{max}]$ are the transaction amount range and the transaction time range of all the transactions belonging to the current product category; $count\_r$ and sum_r denote the aggregates over these transactions; $pointer$ points to its child, which is an R-node or an I-node. Therefore, an R-node contains multiple product categories represented by corresponding records, and these product categories are on the same layer within the product category hierarchy. All R-nodes form an *N-ary tree*, and we term it as a *C-tree (product* ***C****ategory-tree)*.

6.1.2. The MK-tree. In addition to a *C-tree* consisting of R-nodes, following Step 3 in the extension process, each record in an R-node at the brand-based product category

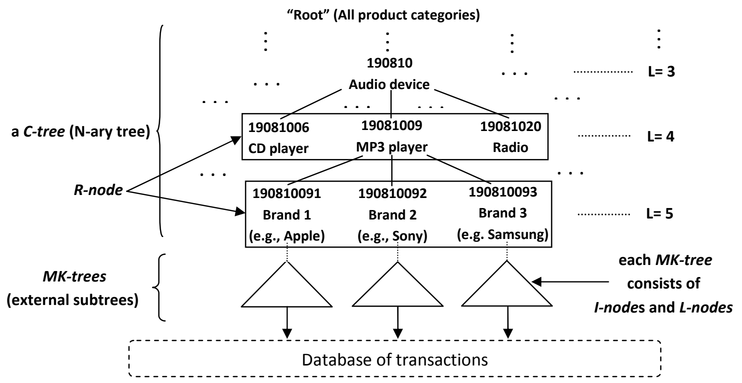


Fig. 9. The structure of our proposed CMK-tree

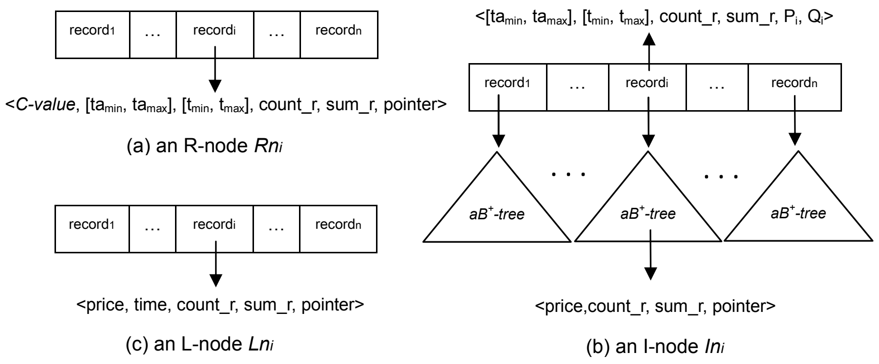

record1
recordi
recordn
<C-value, [tamin, tamax], [tmin, tmax], count_r, sum_r, pointer>
(a) an R-node Rni
<[tamin, tamax], [tmin, tmax], count_r, sum_r, Pi, Qi>
aB+-tree
<price,count_r, sum_r, pointer>
(b) an I-node Ini
<price, time, count_r, sum_r, pointer>
(c) an L-node Lni

Fig. 10. The node structure of a CMK-tree

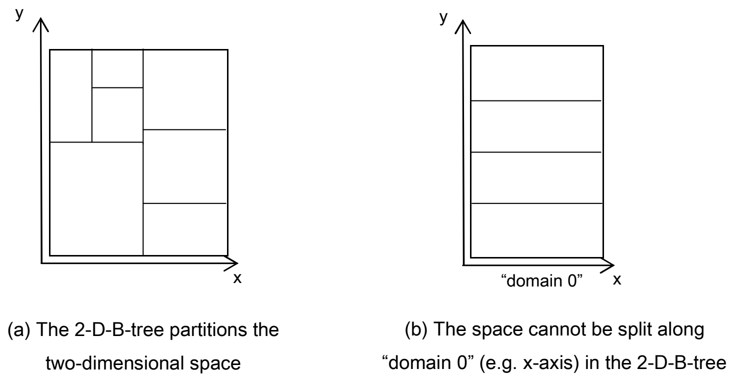

y
x
"domain 0"
(a) The 2-D-B-tree partitions the two-dimensional space
(b) The space cannot be split along "domain 0" (e.g. x-axis) in the 2-D-B-tree

Fig. 11. A special case of 2-D-B-tree

layer (i.e., the bottom of the *C-tree*) points to a subtree that is external to the *C-tree*. Specifically, the design of each such subtree is based on extending the original two-dimensional *K-D-B-tree* or *2-D-B-tree* [Robinson 1981] that is used for indexing spatial data.

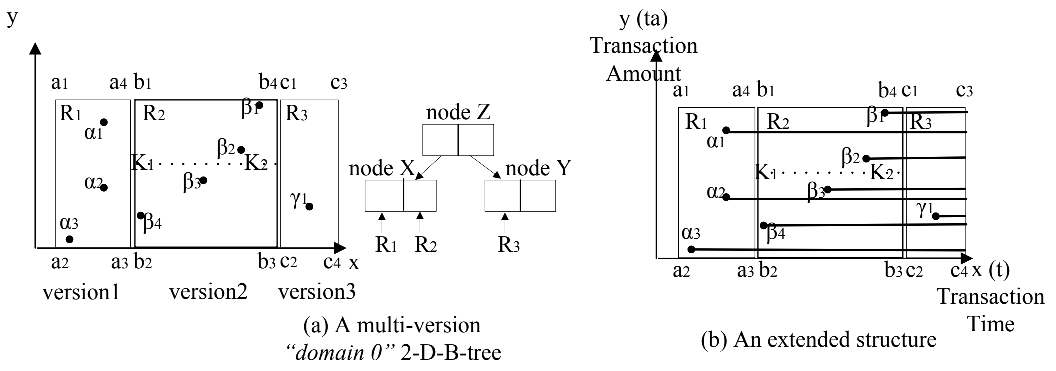


Fig. 12. MK-tree – An extended multi-version "domain 0" two-dimensional K-D-B-tree

The *2-D-B-tree* partitions a two-dimensional space into multiple nonintersecting rectangles (see Fig. 11(a)). Each record in a node in the *2-D-B-tree* corresponds to a rectangular space. Unlike the general structure depicted in Fig. 11(a), a special case *"domain 0" K-D-B-tree* has been proposed in [Robinson 1981]. In particular, for a general *K-D-B-tree*, a rectangular space can be split along any dimension (e.g., x-axis or y-axis). By contrast, for a *"domain 0" K-D-B-tree*, the space cannot be split along a specific dimension. For example, in Fig. 11(b), instead of dividing x-axis, the split can only be operated along the y-axis.

In the *CMK-tree*, the idea of *"domain 0" K-D-B-tree* [Robinson 1981] is adopted to generate each subtree to extend the *C-tree*. Since the x-axis (Transaction Time dimension) continuously moves to the right in our targeted problem, each subtree can be considered as a multi-version structure that makes *partial persistence*[7] [Zhang et al. 2008] a *"domain 0" 2-D-B-tree*. In order to further demonstrate the structure of a subtree, we assume that each point in a two-dimensional space depicted in Fig. 12(a) represents a transaction, and all the transactions belong to the same brand-based product category. Correspondingly, in the subtree generated by these transactions (see Fig. 12(a)), a record in the node X surrounds the rectangular $a_1a_2a_3a_4$ (R_1). This is the first version of *"domain 0" 2-D-B-tree* as illustrated in Fig. 11(b). Another record in the node X surrounds the rectangular $b_1b_2b_3b_4$ (R_2) which is the second version. The record in the node Z at a higher level surrounds a larger rectangular space formed by $a_1a_2b_3b_4$. Furthermore, like the transformation given in Section 2.4.2, each transaction will generate an interval along the Transaction Time dimension (see Fig. 12(b)). As an extended structure, each record simultaneously maintains the transactions whose generated intervals intersect with the left border of the corresponding rectangle as well as the aggregates of transactions. For example, in Fig. 12(b), the record surrounding the rectangle $b_1b_2b_3b_4$ also stores the aggregates of transactions α_1, α_2 and α_3 whose generated intervals along the Transaction Time dimension intersect with the left border b_1b_2 as well as the indexes of these three transactions for computing the trustworthiness of the seller in selling a specific product. To facilitate discussion, we term such a subtree, i.e., an extended ***M**ulti-version "domain 0" **K**-D-B-tree*, as *MK-tree*.

Next, we introduce the I-nodes and the L-nodes of an *MK-tree*. Based on the above description, the record in an I-node In_i (see Fig. 10(b)) has the form

$$< [ta_{min}, ta_{max}],\ [t_{min}, t_{max}],\ count\_r,\ sum\_r,\ P_i,\ Q_i > ,$$

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

where $[ta_{min}, ta_{max}]$ and $[t_{min}, t_{max}]$ surround each nonintersecting rectangle; the term P_i is a pointer pointing to its child, which is an I-node or an L-node; the term Q_i is another pointer pointing to an *aB$^+$-tree* that derives from the *B$^+$-tree* [Bayer and McCreight 1972]. As stated before, the purpose of building an *aB$^+$-tree* is to index the transactions whose generated intervals along the Transaction Time dimension intersect with the left border of the rectangle surrounded by $[ta_{min}, ta_{max}]$ and $[t_{min}, t_{max}]$. In the meantime, the aggregates over these transactions are maintained in $count\_r$ and sum_r. Specifically, each *aB$^+$-tree* is built in the separate transaction amount space. Like the *B$^+$-tree*, the records in an *aB$^+$-tree* are kept sorted based on their transaction amount (price). Also, each node in an *aB$^+$-tree* has the same structure that consists of multiple records, each of which has the form $< price, count\_r, sum\_r, pointer >$ (see Fig. 10(b)). The $pointer$ points to its child, but for the records at leaf level, $pointer$ points to the transaction record stored in the database. Note that each generated *aB$^+$-tree* is based on the transactions in a brand-based product category. In addition, unlike the original *B$^+$-tree*, the number of records to be inserted in an *aB$^+$-tree* may be larger than the number of distinct values of transaction amount (y-coordinates) due to the reason illustrated in *Case 3* in Section 5.

When building an *MK-tree*, in order to avoid duplication of *aB$^+$-trees*, the I-nodes actually include two types: (1) the I-node pointing to *aB$^+$-trees*, and (2) the I-node without pointing to *aB$^+$-trees*. To differentiate the above two situations, we call the I-node '*L-I-node (In(L))*' if its children are L-nodes, and '*I-I-node (In(I))*' otherwise. Therefore, the I-node depicted in Fig. 10(b) is an L-I-node $In(L)_i$. These two node structures will become clearer after introducing insertions in the next subsection.

Each record appearing in an L-node Ln_i (see Fig. 10(c)) also contains $count\_r$ and sum_r because of aggregating the repeated transactions with the same price on a given day which sell the same product. It has the following form

$$< price, time, count\_r, sum\_r, pointer > ,$$

where the *pointer* points to the transaction record stored in the database.

6.2. The Construction of a CMK-tree

This section formally describes the insertion of transactions for the *CMK-tree*, including the path in the product category hierarchy (*C-hrchy$_i$*), transaction amount (ta_i), transaction time (t_i) and the rating for the transaction (r_i). In the meantime, the transaction records are inserted in chronological order.

6.2.1. Insertion. Before inserting the data of a newly happened transaction into a *CMK-tree*, a path is first searched in a *C-tree* from top (the product category root) to bottom (the brand-based product category) (see Fig. 9) based on the *C-hrchy* of the transaction. If the product in the transaction belongs to a new product category on which the seller has no prior transactions, the new records are generated for this product category as well as its sub-categories and inserted to the corresponding R-nodes. Otherwise, the set of ranges and aggregates (i.e., $[ta_{min}, ta_{max}]$, $[t_{min}, t_{max}]$, $count\_r$, sum_r) maintained in each record along the path are updated accordingly. After that, the insertion operations should be performed in an *MK-tree* pointed by the corresponding record in an R-node at the brand-based product category layer. Fig. 13 depicts the state of a *CMK-tree* after inserting the data of three transactions trading different products, which belong to the same product category with the same *C-hrchy*. In addition, the information $< ta_i$ (transaction amount), t_i (transaction time), r_i (rating) $>$ of three transactions is $< 5, 1, 1 >$, $< 15, 1, 1 >$ and $< 10, 2, 1 >$.

6.2.2. Split. The split of an R-node is relatively simple. Unless stated otherwise, in the following example, we assume the capacity of all the nodes is five. In addition,

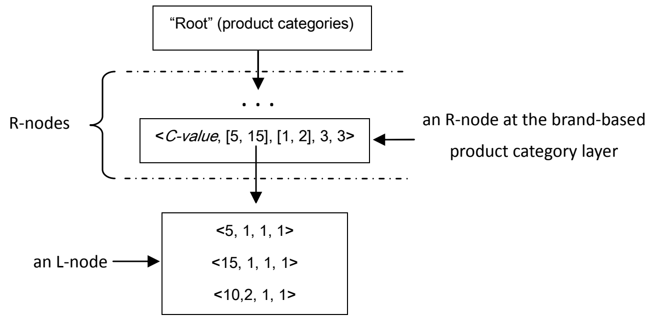

"Root" (product categories)
R-nodes
<C-value, [5, 15], [1, 2], 3, 3>
an R-node at the brand-based product category layer
an L-node
<5, 1, 1, 1>
<15, 1, 1, 1>
<10,2, 1, 1>

Fig. 13. The state of a CMK-tree after inserting three transactions

the field sum_r in all the records is ignored, and we only use the field $count\_r$ as the example to illustrate the aggregation process. Fig. 14(a) shows an example of the R-node split where Rn_1 is an R-node containing a record $<$ *C-value*$_p, [10, 60], [1, 5], 32 >$ at a higher level of the *C-tree*. The R-node Rn_1 points to its child node Rn_2 containing five records, i.e., from *C-value*$_{ch_1}$ to *C-value*$_{ch_5}$. Here we use *C-value*$_p$ and *C-value*$_{ch}$ to denote the parent product category and the child product category, respectively. $[10, 60]$ is the transaction amount range, and $[1, 5]$ is transaction time range. 32 is the total number of transactions in the current product category represented by *C-value*$_p$.

When a new record $<$ *C-value*$_{ch_6}, [15, 15], [6, 6], 1 >$ (i.e., a new product category) is inserted to the R-node Rn_2, it overflows, and then this record is moved to a new R-node Rn_3. In the meantime, Rn_3 is pointed to by another new record generated in Rn_1, and the data fields $[ta_{min}, ta_{max}]$, $[t_{min}, t_{max}]$, $count\_r$ and sum_r in this record are updated to reflect the ranges and aggregates of its child node (see Fig. 14(a)).

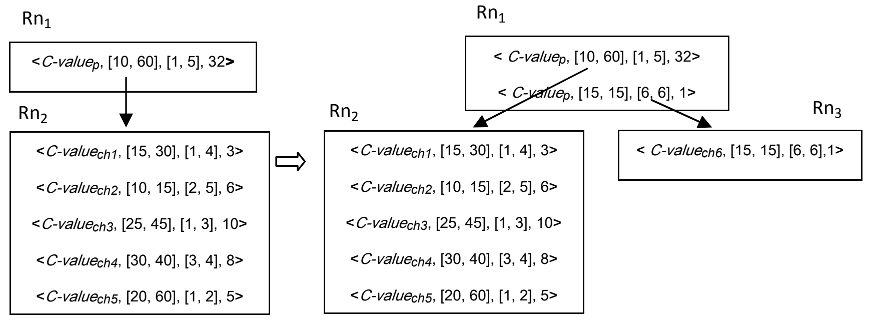
The split of either an L-node or an I-node that occurs in an *MK-tree* is more complicated, and includes two situations, respectively.

1) L-node split

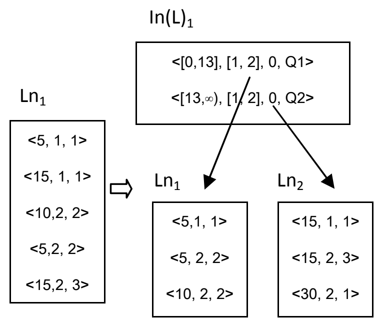
Situation 1: If the record to be inserted in an L-node has the same transaction time (i.e., x-coordinate) as a record existing in it, the L-node splits according to the transaction amounts of all the records it contains.

Fig. 14(b) illustrates the split of an L-node Ln_1 after inserting a new record $< 30, 2, 1 >$, which leads to a new L-I-node $In(L)_1$. The number 30 is the *transaction amount*, 2 is the *transaction time* and 1 is the field $count\_r$ in sequence. Note that the inserted record first needs to be checked whether the transactions with the same price selling the same product have already been indexed by the records in Ln_1. If there exists repeated transactions, instead of splitting the L-node Ln_1, it is only to update $count\_r$ and sum_r in the corresponding record within Ln_1. For the L-I-node $In(L)_1$, in order to get the full partition on the entire transaction amount space, the boundary is set to the intermediate value of the maximal transaction amount in the new L-node Ln_1 and the minimal transaction amount in the L-node Ln_2. For instance, in Fig. 12(b), the y-coordinate of a boundary K_1K_2 equals to $\lfloor \frac{\mathbf{y}_{\beta_2}+\mathbf{y}_{\beta_3}}{\mathbf{2}} \rfloor$. Hence, the two records in a generated L-I-node $In(L)_1$ are $< [0, 13], [1, 2], 0, Q_1 >$ and $< [13, \infty), [1, 2], 0, Q_2 >$, respectively, where both Q_1 and Q_2 point to an *aB*$^+$*-tree*.

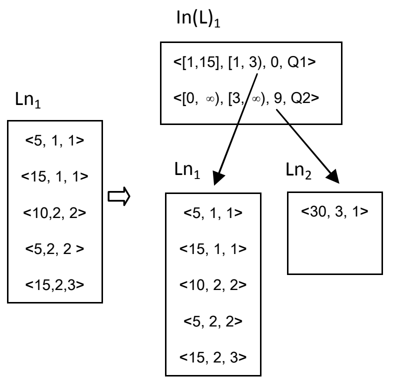
— ***aB*$^+$*-tree split*:** When an L-node pointed by a record in the L-I-node is split into two L-nodes, correspondingly, the *aB*$^+$*-tree* pointed by this record also needs to split. For example, we assume that all the transactions represented by the points in Fig. 12(b) are in the same brand-based product category. If the rectangle $b_1b_2b_3b_4$ splits into


Rn1
<C-valuep, [10, 60], [1, 5], 32>
Rn2
<C-valuech1, [15, 30], [1, 4], 3>
<C-valuech2, [10, 15], [2, 5], 6>
<C-valuech3, [25, 45], [1, 3], 10>
<C-valuech4, [30, 40], [3, 4], 8>
<C-valuech5, [20, 60], [1, 2], 5>
Rn1
< C-valuep, [10, 60], [1, 5], 32>
< C-valuep, [15, 15], [6, 6], 1>
Rn2
<C-valuech1, [15, 30], [1, 4], 3>
<C-valuech2, [10, 15], [2, 5], 6>
<C-valuech3, [25, 45], [1, 3], 10>
<C-valuech4, [30, 40], [3, 4], 8>
<C-valuech5, [20, 60], [1, 2], 5>
Rn3
< C-valuech6, [15, 15], [6, 6],1>

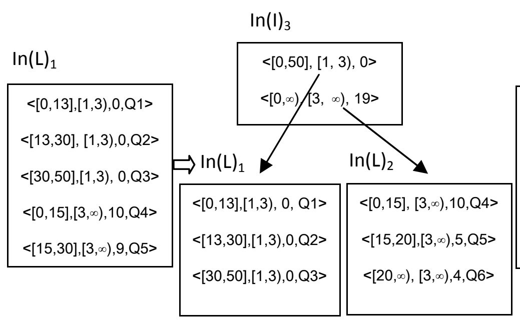
(a) The split of an R-node



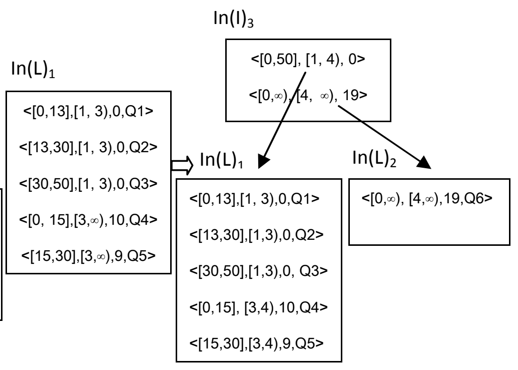
(b) The split of an L-node in Situation 1



(c) The split of an L-node in Situation 2



(d) The split of an I-node in Situation 1



(e) The split of an I-node in Situation 2

Fig. 14. The construction of a CMK-tree

two rectangles $b_1K_1K_2b_4$ and $K_1b_2b_3K_2$, the original *aB$^+$-tree* splits into two *aB$^+$-trees* that store the information of one point (i.e., α_1) and two points (i.e., α_2, α_3) in $a_1a_2a_3a_4$, respectively.

Situation 2: If the record to be inserted in an L-node has a different transaction time (i.e., x-coordinate), a new L-node is generated.

Fig. 14(c) illustrates that if the record inserted to the L-node Ln_1 is $< 30, 3, 1 >$, a new L-node Ln_2 is generated. In the meantime, an L-I-node $In(L)_1$ is generated with two records pointing to Ln_1 and Ln_2. The two records in the I-node $In(L)_1$ are $< [1, 15], [1, 3), 0, Q_1 >$ and $< [0, \infty), [3, \infty), 9, Q_2 >$.

— ***Generate a new aB$^+$-tree*:** In Fig. 14(c), the record $< [0, \infty), [3, \infty), 9, Q_2 >$ in the I-node $In(L)_1$ surrounds a new rectangle. Accordingly, a new *aB$^+$-tree* pointed by Q_2 has to be generated to index the transactions in the same brand-based product category whose generated intervals along the Transaction Time dimension intersect with the left border $3 : [0, \infty)$. In addition, the number 9 (i.e., $count\_r$) denotes the total number of such transactions.

In Fig. 14(c), we assume that two records $< 5, 1, 1 >$ and $< 5, 2, 2 >$ in the L-node Ln_1 index the transactions selling the same product but traded at different time; another two records $< 15, 1, 1 >$ and $< 15, 2, 3 >$ index the transactions selling different products but traded at the same price. The new *aB$^+$-tree* pointed by Q_2 is equivalent to the *aB$^+$-tree* pointed by Q_1 after inserting four different records: $< 5, 3 >$, $< 10, 2 >$, $< 15, 1 >$ and $< 15, 3 >$. However, since the record $< [1, 15], [1, 3), 0, Q_1 >$ in the I-node $In(L)_1$ represents the initial rectangle, the *aB$^+$-tree* pointed by Q_1 is null in this example and the field $count\_r$ is 0. Here the above four records do not include the Transaction Time dimension, as each *aB$^+$-tree* is built in a separate Transaction Amount dimension. The insertion and split of an *aB$^+$-tree* are the same as those of a *B$^+$-tree* [Bayer and McCreight 1972].

Notice that 3 in the record $< 5, 3 >$ is the aggregated value for the field of $count\_r$ in two records $< 5, 1, 1 >$ and $< 5, 2, 2 >$, as they index the transactions selling the same product. Also, the records $< 15, 1 >$ and $< 15, 3 >$ index the transactions selling different products, and thus they should be inserted separately.

— ***Merge aB$^+$-trees*:** Several *aB$^+$-trees* may need to be merged to generate a new *aB$^+$-tree*. In Fig. 14(b), assume another record $< [0, \infty), [3, \infty), 10, Q_3 >$ is to be inserted in the L-I-node $In(L)_1$. Since two records $< [0, 13], [1, 2], 0, Q_1 >$ and $< [13, \infty), [1, 2], 0, Q_2 >$ in the original L-I-node $In(L)_1$ have the same time range $[1, 2]$, the two *aB$^+$-trees* pointed by Q_1 and Q_2 respectively first need to be merged to form a new *aB$^+$-tree*. Then, the newly generated *aB$^+$-tree* is pointed by Q_3. The above operations are performed on all the records with the same time range.

Here we need to emphasize that since the L-node split in Situation 2 leads to the problem of space utilization, in order to guarantee the minimum space utilization for each L-node larger than 50%, Situation 2 happens only when all the L-nodes are at least half full.

2) I-node split

Situation 1: If the record to be inserted in an I-node has the same transaction time range as an existing record in that I-node, all the records with the same transaction time range as the inserted record will be moved to a newly generated I-node. In addition, if each record in an I-node has the same transaction time range as the inserted record, the same as the L-node split in Situation 1, the I-node splits according to the transaction amount ranges of all the records it contains.

In Fig. 14(d), for example, if the record $< [15, 30], [3, \infty), 9, Q_5 >$ in the L-I-node $In(L)_1$ splits into two new records $< [15, 20], [3, \infty), 5, Q_5 >$ and $< [20, \infty), [3, \infty), 4, Q_6 >$, the above two records lead to the overflow of $In(L)_1$. Then, the node $In(L)_1$ will continue to split into two L-I-nodes: a new $In(L)_1$ and a new $In(L)_2$. In the meantime, another I-I-node $In(I)_3$ with two records $< [0, 50], [1, 3), 0 >$ and $< [0, \infty), [3, \infty), 19 >$ ($19 = 10 + 9 = 10 + 5 + 4$) is generated to point to the node $In(L)_1$ and the node $In(L)_2$, respectively.

Situation 2: If the record to be inserted in an I-node has a different transaction time range, similar to the L-node split in Situation 2, a new I-node is generated to contain the inserted record. Such a strategy is to guarantee the maximum space utilization for an I-node.

In Fig. 14(e), after the record $< [0,\infty), [4,\infty), 19 >$ inserted to $In(L)_1$, a new L-I-node $In(L)_2$ is generated. In the meantime, an I-I-node $In(I)_3$ with two records $< [0,50], [1,4), 0 >$ and $< [0,\infty), [4,\infty), 19 >$ is generated to point to the I-nodes $In(L)_1$ and $In(L)_2$, respectively. Notice that each record in node $In(I)_3$ will not include a pointer Q_i so as to avoid duplication of the *aB*$^+$*-tree*. This is because the same *aB*$^+$*-tree* has already been indexed by its child node $In(L)_2$, which is an L-I-node.

Algorithm 1 presents pseudo-code for the complete insertion process.

ALGORITHM 1: The CMK-tree Construction

```
Input: A transaction $TR_i$ includes C-hrchy$_i$, $ta_i$, $t_i$ and $r_i$.
Output: CMK-tree
 1: // construct a C-tree
 2: Starting from the "Root" of the product category hierarchy
 3: Determine the path in C-tree based on the C-hrchy$_i$ of each transaction $TR_i$.
 4: for all R-nodes along the path do
 5:     if The product traded in $TR_i$ belongs to a new product category on which the seller has no prior transactions then
 6:         i. insert the generated new record in corresponding R-node.
 7:         ii. if the R-node overflows, split.
 8:     else
 9:         update corresponding ranges and aggregates maintained in a record
10:     end if
11: end for
12: // construct the MK-trees that are external to the C-tree
13: for all L-nodes and I-nodes in the path from bottom up do
14:     if the node is the L-node then
15:         i. if transaction time $t_i$ is different from any transactions in the L-node, and this L-node is at least half full,
            then generate a new L-node.
16:         ii. if transaction time $t_i$ is the same as a record in the L-node, and repeated transactions have already been
            indexed in this L-node, then update corresponding $count\_r$ and $sum\_r$.
17:         iii. otherwise, insert the transaction $TR_i$
18:         iv. if the L-node overflows, split
19:     else if the node is the I-node then
20:         i. insert the generated new records
21:         ii. if the I-node overflows, split
22:     end if
23: end for
```

6.3. CTT Computation Algorithm

Basically, the CTT computation algorithm answers a buyer's typical CTT queries covering three transaction dimensions based on our proposed *CMK-tree*. The processing of CTT computation starts by locating product category in the *C-tree* according to the *C-value* (i.e., product category) in a buyer's query. Then, it computes the left border VRA and the right border VRA (see Fig 4(b)) in one or several *MK-trees*, respectively, depending on the number of brand-based product categories that are included in a CTT query.

For example, to answer a typical CTT query: <product-category: "*Audio device*", price-range: "\$100-\$200", time-range: "*the latest 6 months*" >, all the sub-categories of the product category specified in a CTT query are first considered (see Fig. 9). As each record in an R-node contains a transaction amount range ($[ta_{min}, ta_{max}]$) and a transaction time range ($[t_{min}, t_{max}]$) for its corresponding product category, there are three cases that should be differentiated.

Case 1: *If a CTT query on the transaction amount range and the transaction time range falls into the region surrounded by* $[ta_{min}, ta_{max}]$ *and* $[t_{min}, t_{max}]$*, then it is not necessary to search its child node (sub-categories). Instead,* $count\_r$ *and* sum_r *in that record can be used directly.*

Case 2: *If a CTT query on the transaction amount range and the transaction time range are out of the region surrounded by* $[ta_{min}, ta_{max}]$ *and* $[t_{min}, t_{max}]$*, then it is not necessary to search its child node (sub-categories) either.*

Case 3: *If a CTT query on the transaction amount range and the transaction time range overlaps with the region surrounded by* $[ta_{min}, ta_{max}]$ *and* $[t_{min}, t_{max}]$*, then the search iteratively executes from Case 1 to Case 3 in its descendants until reaching the layer of I-nodes. Taking each reached I-node as the root of an MK-tree, all the corresponding MK-trees are searched for the left border VRA and the right border VRA, respectively.*

For *Case 3*, we take the computation of a VRA $2 : [0, 5]$ as an example to introduce the search process in an *MK-tree*. As depicted in Fig. 14(d), the I-nodes, the rectangle represented by which contains $2 : [0, 5]$, are iteratively searched until reaching the layer of L-I-nodes, and thus the record $< [0, 13], [1, 3), 0, Q_1 >$ in the L-I-node $In(L)_1$ is selected. In order to compute the VRA $2 : [0, 5]$, both the *aB*$^+$*-tree* and the L-node pointed by the above record need to be searched, and the VRA equals to the sum of the two search results. Note that instead of visiting only one record as introduced in the above example, the search for computing the VRA may be executed on several *aB*$^+$*-trees* as well as L-nodes pointed by the corresponding records, respectively, depending on the number of the rectangles overlapped by the query range in the Transaction Amount dimension. For instance, to compute another VRA $3 : [10, 30]$ based on Fig. 14(d), two records $< [0, 15], [3, \infty), 10, Q_4 >$ and $< [20, \infty], [3, \infty), 4, Q_6 >$ in the L-I-node $In(L)_2$ are selected for conducting the further searches. The aggregation results ($count\_r$ and sum_r) in the record $< [15, 20], [3, \infty), 4, Q_5 >$ can be used directly, as the transaction amount range $[15, 20]$ in that record falls into the query range $[10, 30]$ in the Transaction Amount dimension.

If the transaction time for a VRA equals to the left border of a rectangle represented by the selected record in an L-I-node, search only performs on the *aB*$^+$*-tree* pointed by this record. For example, in Fig. 14(e), to compute the VRA $\mathbf{3} : [5, 10]$, the record $< [0, 15], [\mathbf{3}, 4), 10, Q_4 >$ in the L-I-node $In(L)_1$ is selected. Instead of searching L-node, only the *aB*$^+$*-tree* pointed by the above record is searched. However, since the right border in CTT computation is always fixed to the point "*now*", the search for computing the right border VRA is performed on both the *aB*$^+$*-trees* and L-nodes pointed by the selected records.

Algorithm 2 describes the process of CTT computation in a *CMK-tree*.

6.4. Structure and Performance Analysis

In this section, we will provide an analytical study on the *CMK-tree*, focusing on its structure and query performance. The symbols and their meanings used in our analysis are explained in Table II.

Property 1. *Each MK-tree in a CMK-tree represents a two-dimensional space formed by Transaction Amount and Transaction Time for all the transactions in a brand-based product category. All the records in each layer of I-nodes in an MK-tree fully partition the corresponding two-dimensional space into multiple nonintersecting rectangles.*

Assume that all the transactions in a brand-based product category are represented by a number of points in two-dimensional space depicted in Fig. 12(a). Since the insertions come in the nondecreasing time order, an new insertion only happens in the latest version of *"domain 0" 2-D-B-tree*, for example, the version 3 in Fig. 12(a). In

ALGORITHM 2: CTT Computation Algorithm in a CMK-tree

```
Input: A typical CTT query with specific product category, transaction amount range ($[ta_1, ta_2]$) and transaction time
    range ($[t_1, t_2]$).
Output: CTT value
 1: CTT=0
 2: $count\_r_1$=0, $sum\_r_1$=0, $count\_r_2$=0, $sum\_r_2$=0
 3: The Searching starts from the "Root" of product category hierarchy
 4: Determine the layer in product category hierarchy based on the CTT query on product category, and return
    corresponding record R in an R-node.
 5: begin Search(CMK-tree, R)
 6: if the record is in an R-node then
 7:     Case 1: CTT=CTT+$\frac{sum\_r}{count\_r}$, $count\_r$ and $sum\_r$ are from the record
 8:     Case 2: CTT=CTT
 9:     Case 3: Search(CMK-tree, R$_{childnode}$)
10: else if the record is in an I-node then
11:     Let rec$_1$ be the index record whose rectangle contains $t_1 : [ta_1, ta_2]$ // for the left border VRA
12:     Let rec$_2$ be the index record whose rectangle contains $t_2 : [ta_1, ta_2]$ // for the right border VRA
13:     while neither rec$_1$ nor rec$_2$ is a record in L-I-nodes do
14:         i. rec$_1$ be its child whose rectangle contains $t_1 : [ta_1, ta_2]$
15:         ii. rec$_2$ be its child whose rectangle contains $t_2 : [ta_1, ta_2]$
16:     end while
17:     for all rec$_1$s do
18:         if $t_1$ equals to the left border of rec$_1$ then
19:             only search aB$^+$-tree that is pointed by rec$_1$ for left border to aggregate $count\_r_1$ and $sum\_r_1$
20:         else
21:             the search conducts on both the aB$^+$-tree and the L-node pointed by rec$_1$ to aggregate $count\_r_1$ and
            $sum\_r_1$
22:         end if
23:     end for
24:     for all rec$_2$s do
25:         the search conducts on both aB$^+$-tree and L-node pointed by rec$_2$ to aggregate $count\_r_2$ and $sum\_r_2$
26:     end for
27:     CTT=CTT+$\frac{sum\_r_2 - sum\_r_1}{count\_r_2 - count\_r_1}$
28: end if
29: end Search
```

Table II. List of symbols

symbol	explanation
S	a seller
$Trans$	the transaction set for the seller S
n	the number of past transactions contained in $Trans$
m	the number of brand-based product categories (see Fig. 8) to which n past transactions belong
n_i	the number of points in a two-dimensional space formed by the transactions in a brand-based product category.
h	the height of the product category hierarchy for a newly happened transaction
n_h	the number of points in a two-dimensional space formed by the transactions in $Trans$ which are in the same brand-based product category as the newly happened transaction
nc_L	the capacity of an L-node in a *CMK-tree*
nc_I	the capacity of an I-node in a *CMK-tree*

Note:
m*:* The number of brand-based product category determines the size of *C-tree*
n_i*:* For m brand-based product categories, we have $\sum_{i=1}^{m} n_i \leq n$. This is because one point may represent a set of repeated transactions that occurred on a day (see *Case 2* in Section 5). In practice, $\sum_{i=1}^{m} n_i$ may be much less than n.
nc_I*:* The difference of node capacity between I-I-node and L-I-node is ignored in our discussion.

the meantime, the division within each version can only be operated along the Transaction Amount dimension (y-axis). As shown in Fig. 14(b) and Fig. 14(c), each record in an L-I-node corresponds to either a new version for *"domain 0" 2-D-B-tree* (e.g., $< [0, \infty), [3, \infty), 9, Q_2 >$) or a partition in the latest version (e.g, $< [0, 13], [1, 2], 0, Q_1 >$). Obviously, these records partition the complete two-dimensional space. More importantly, the rectangles formed by them are nonintersecting. In addition, except the records in L-I-nodes, the records in I-I-nodes (a higher level) still fully partition the two-dimensional space into multiple nonintersecting rectangles, for example, the records in node $In(I)_3$ as depicted in Fig. 14(d) and Fig. 14(e).

Property 2. *To answer a CTT query for each brand-based product category, the CMK-tree delivers almost linear query performance.*

Let us go back to the CTT computation algorithm given in Section 6.3, where the method of computing the left border VRA and the right border VRA is adopted while answering a CTT query. In order to clearly understand property 2, we first examine the performance of computing each VRA. Property 1 has illustrated that the transaction amount ranges and transaction time ranges of the records in the I-nodes in an *MK-tree* do not intersect each other. Thus, when computing a VRA, the search traverses from top to bottom in an *MK-tree* until reaching the layer of L-I-nodes. Then, one or several corresponding records in L-I-nodes are chosen. Finally, both the *aB$^+$-trees* and the leaf nodes pointed to by these records are searched. To sum up, the structure of *MK-tree* achieves logarithmic time cost ($O(\log n)$) for computing each VRA. Hence, to answer a buyer's CTT queries for a specific brand-based product category, the query of *CMK-tree* is almost linear. Also, this property has been demonstrated in the experiments, the results of which are to be introduced in Section 8.

When a buyer performs *"roll up"* operations, the search iteratively performs from *Case 1* to *Case 3* introduced in Section 6.3 in the descendants of the current R-node, and finally one or several *MK-trees* are selected. The search then continues in the s-elected *MK-trees* twice for computing the left border VRA and the right border VRA, respectively. Note that the number of *MK-trees* may be much less than m in practice. Therefore, the process has the linearithmic time cost ($O(n \log n)$) in total. However, the *CMR-tree* has better performance than all three existing approaches that were proposed in [Zhang et al. 2014] (see Section 8). This is a significant advantage in answering CTT queries.

Next, we analyse the space utilization of the *CMK-tree* that is important to evaluate disk-based index schemes. We adopt the same analysis method as in [Kang et al. 2004] and consider the predictability of space utilization, i.e., minimum space utilization of each node.

Lemma 1. *The minimum space utilization is no less than $\frac{nc_L}{2}$ for an L-node; The minimum space utilization is no less than $\frac{nc_I}{3}$ on average for an I-node.*

Proof. Let t_1, t_2 and t_3 be three different time periods. Suppose an initial state that all the records in the L-node Ln_1 have the same transaction time t_1. For a new record with the transaction time t_2 to be inserted in Ln_1, there are two cases. (1) If space utilization of Ln_1 is no less than $\frac{nc_L}{2}$, a new L-node Ln_2 is established. (2) Otherwise, the new record with the transaction time t_2 is inserted into Ln_1 until it overflows. Then, the node Ln_1 splits into a new Ln_1 and a new L-node Ln_2. The space utilization of each generated L-node is still no less than $\frac{nc_L}{2}$. All the records in the two L-nodes are within the same time range $[t_1, t_2]$. In this case, if the new records with the transaction time t_2 continue to be inserted in an L-node, either Ln_1 or Ln_2 is selected as the targeted L-node depending on the transaction amount of the new record. Note that both Ln_1 and

Ln_2 might split again during insertion, but the minimum space of any generated new L-nodes is no less than $\frac{nc_L}{2}$, and the records maintained in each node are within a time range $[t_1, t_2]$. The above operations are repeated until a record with the transaction time t_3 is inserted. This is because a new L-node is established for that record. So, the minimum space utilization is no less than $\frac{nc_L}{2}$ for an L-node. In fact, except the L-nodes with the minimum space utilization no less than $\frac{nc_L}{2}$, there is at most one L-node with the space utilization less than $\frac{nc_L}{2}$. In particular, this specific L-node includes the records that are most recently inserted. For example, the newly generated L-node maintains only one record with the transaction time t_3.

To estimate the space utilization of I-nodes, we consider the worst case. Let t_1, t_2, t_3 and t_4 be four different time periods. Still, we suppose an initial state that a full I-node In_1 with nc_I records includes only one record with the transaction time range $[t_1, t_2]$. The rest of the records in In_1 have the same transaction time range $[t_3, t_4]$. If a new record with a transaction time range $[t_3, t_4]$ to be inserted in the I-node In_1, the node In_1 overflows. Then, it splits into a new In_1 maintaining the one record with the transaction time range $[t_1, t_2]$ and another full I-node Ln_2. All the records in node Ln_2 have the same time range $[t_3, t_4]$. In such a case, if a new record with the transaction time range $[t_3, t_4]$ continues to be inserted in an I-node, the node Ln_2 is selected as the targeted I-node. Then, the node In_2 overflows and splits into two I-nodes according to the transaction amount ranges of all the records it contains. Hence, due to continuous splits of I-nodes, the records in the original full I-node In_1 are distributed in three different I-nodes. Therefore, we can conclude that, in the worst case, the minimum space utilization is no less than $\frac{nc_I}{3}$ on average for an I-node. □

Lemma 2. *The height of an MK-tree in the CMK-tree is at most* $\lceil \log_{\lceil \frac{nc_I}{3} \rceil}^{\lfloor \frac{n_i}{\lceil \frac{nc_L}{2} \rceil} \rfloor} \rceil + 1$ $(\forall i \in [1, m])$. *The number of aB*$^+$*-trees in the CMK-tree pointed by L-I-nodes is at most* $\sum_{i=1}^{m} \lfloor \frac{n_i}{\lceil \frac{nc_L}{2} \rceil} \rfloor$.

Proof. First, we examine the height of an *MK-tree* that is built based on the transactions in a brand-based product category. For m *MK-trees* in a *CMK-tree*, we focus on analyzing one of them. Suppose the past transactions in a brand-based product category form n_i $(\forall i \in [1, m])$ points in a two-dimensional space. In Lemma 1, we have proved that the minimum space utilization is no less than $\frac{nc_L}{2}$ for an L-node. In addition, there may exist one L-node with space utilization less than $\frac{nc_L}{2}$. Hence, the number of L-nodes in an *MK-tree* is at most $\lfloor \frac{n_i}{\lceil \frac{nc_L}{2} \rceil} \rfloor$. The above L-nodes will be indexed by I-nodes, each of which has the minimum space utilization no less than $\frac{nc_I}{3}$ on average. Hence, the height of an *MK-tree* in the *CMK-tree* is at most $\lceil \log_{\lceil \frac{nc_I}{3} \rceil}^{\lfloor \frac{n_i}{\lceil \frac{nc_L}{2} \rceil} \rfloor} \rceil + 1$. Since the number of transactions traded by the seller S in m brand-based product categories has an imbalanced distribution, each *MK-tree* has a different height. The *CMK-tree* is an unbalanced tree.

Second, we examine the total number of *aB*$^+$*-trees*, each of which is pointed by a record in an L-I-node. There are at most $\lfloor \frac{n_i}{\lceil \frac{nc_L}{2} \rceil} \rfloor$ L-nodes in an *MK-tree*, and each L-node is pointed by a record in an L-I-node. Thus, the number of *aB*$^+$*-trees* in an *MK-tree* is also $\lfloor \frac{n_i}{\lceil \frac{nc_L}{2} \rceil} \rfloor$ at most. Therefore, for m *MK-trees* in the *CMK-tree*, the total number of *aB*$^+$*-trees* is at most $\sum_{i=1}^{m} \lfloor \frac{n_i}{\lceil \frac{nc_L}{2} \rceil} \rfloor$. □

Theorem 1. *In a CMK-tree, the number of nodes accessed by an insertion operation is* $O(h) + O(\log_{nc_I}^{\frac{n_h}{nc_L}}) + 1$.

Proof. For an insertion operation in the *CMR-tree*, we consider the insertion cost in the worst case. As described in Section 6.2.1, an insertion operation first searches a path in the *C-tree* based on the *C-hrchy* of the newly happened transaction. The number of accessed nodes is $O(h)$. Then, the insertion operation traverses an *MK-tree*. Lemma 2 illustrates that the height of an *MK-tree* in the *CMK-tree* is at most $\lceil \log_{\lceil \frac{nc_I}{3} \rceil}^{\lfloor \frac{n_i}{\lceil \frac{nc_L}{2} \rceil} \rfloor} \rceil + 1$ $(\forall i \in [1, m])$. Hence, the number of accessed nodes for inserting a newly happened transaction in an *MK-tree* is $O(\log_{nc_h}^{\frac{n_h}{nc_L}}) + 1$. As a result, the total number of accessed nodes is $O(h) + O(\log_{nc_I}^{\frac{n_h}{nc_L}}) + 1$ for an insertion operation. □

6.5. Extending the CMK-tree for High-Dimensional CTT Computation

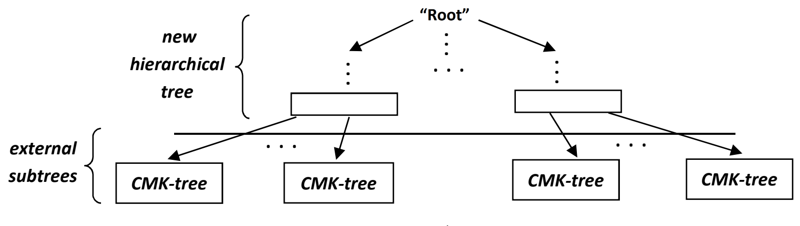
As we have mentioned before, the state-of-the-art trust and reputation management approaches consider single context and lack dimensionality. By contrast, our proposed *ReputationPro* model targets the multi-context transaction trust computation problem in real-world e-commerce environments taking multiple transaction dimensions into account. Basically, our proposed *CMK-tree* focuses on CTT computation with three identified important transaction context dimensions: Product Category, Transaction Amount and Transaction Time. More specifically, in *CMK-tree*, each *MK-tree* maintains the aggregations of trust ratings that incorporate two dimensions of Transaction Amount and Transaction Time. Then, multiple *MK-trees* are adopted as the subtrees to extend a *C-tree*. Thus the *C-tree* maintains the aggregations of trust ratings that incorporate all three identified transaction context dimensions. Next, considering the possibility to take extra dimension(s) into account in CTT computation, we propose *CMK$^+$-tree* which extends the *CMK-tree*. In the following, we first consider the situation with the additional fourth dimension.

6.5.1. CMK$^+$-tree. With regard to a new dimension, as pointed out in Section 3, we need to identify its characteristic as either a hierarchical dimension (e.g., similar to the Product Category dimension) or a linear dimension (e.g., similar to the Transaction Amount dimension). Accordingly, two algorithms are proposed to construct the *CMK$^+$-tree*. In order to differentiate these two algorithms, we denote the generated *CMK$^+$-tree* as *CMK$^+$-tree(HD)* if the fourth dimension is a Hierarchical Dimension, or *CMK$^+$-tree(LD)* if the fourth dimension is a Linear Dimension.

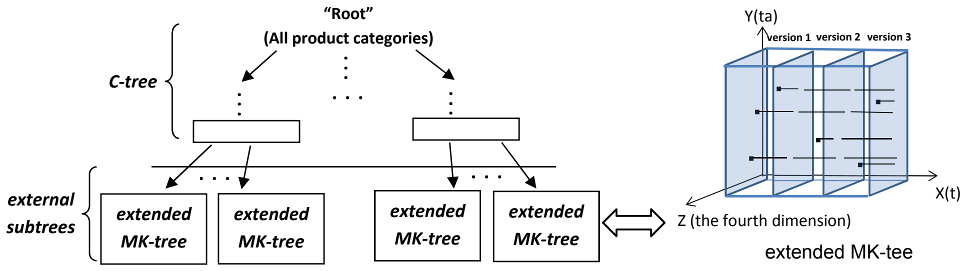
Algorithm 3 (*CMK$^+$-tree(HD)* construction): The construction of *CMK$^+$-tree(HD)* is to adopt multiple *CMK-trees* as subtrees, which cover three transaction context dimensions (i.e., Product Category, Transaction Amount and Transaction Time) to extend the new hierarchical tree, i.e., the fourth dimension (see Fig. 15(a)):

- When inserting a record in a *CMK$^+$-tree(HD)*, following the search process in a *C-tree* as depicted in Algorithm 1, a path is located from top to bottom in the new hierarchical tree. In the meantime, the update operations are performed in the corresponding records along that path. Different from the *C-tree* in the *CMK-tree*, each record in the new hierarchical tree includes aggregations of trust ratings that cover all the four transaction context dimensions. Therefore, when updating along the path, the corresponding aggregations covering four dimensions are updated.
- When a leaf node in the hierarchical tree is reached, Algorithm 1 is called to generate a *CMK-tree* as a subtree to extend the new hierarchical tree.

Algorithm 4 (*CMK$^+$-tree(LD)* construction): Similar to the *CMK-tree*, the construction of a *CMK$^+$-tree(LD)* is to adopt multiple subtrees to extend a *C-tree*. However, different from the original *C-tree*, the *C-tree* in the *CMK$^+$-tree(LD)* includes aggrega-



(a) CMK$^+$-tree(HD)



(b) CMK$^+$-tree(LD)

Fig. 15. The construction of the *CMK$^+$-tree*

tions of trust ratings that cover four transaction context dimensions. In addition, as shown in Fig. 15(b), each subtree in a *CMK$^+$-tree(LD)* maintains the aggregates of ratings covering three linear dimensions: Transaction Amount, Transaction Time and the fourth dimension:

- When inserting a record in a *CMK$^+$-tree(LD)*, following Algorithm 1, a path is located in a *C-tree* from top to bottom based on the *C-hrchy* (see Section 4.1) of a transaction record. In the meantime, the corresponding records are updated along that path, where the aggregations maintained in a record cover four dimensions.
- When a leaf node in the *C-tree* is reached, an extended *MK-tree* is built as a subtree to extend the *C-tree*. Essentially, the structure of the extended *MK-tree*, as shown in Fig. 15(b), is an extended multi-version *"domain 0" 3-D-B-tree*. In the literature, the *R-tree* based variants and the *K-D-B-tree* based variants are two popular structures which can be used for indexing data in any linear dimensionality [Böhm et al. 2001]. On the other hand, as illustrated in Section 6.1.2, the multi-version structure is an effective means to support aggregation operations along a temporal dimension. Therefore, correspondingly, there are two different ways to generate each subtree: 1) multi-version *aR-tree* [Tao et al. 2004] and, 2) extended multi-version *"domain 0" K-D-B-tree*. Since the multi-version *aR-tree* belongs to a variant of the *R-tree*, its performance is still subject to the number of overlapped MBRs (see Section 2.4.1). By contrast, our proposed *MK-tree* is an extended multi-version *"domain 0" 2-D-B-tree* which has been proved to be efficient for CTT computation (see Section 6.4). Following the idea of *MK-tree* depicted in Section 6.1.2, we can continuously extend it to form an extended multi-version *"domain 0" 3-D-B-tree*.

***CMK$^+$-tree* based CTT computation:** Now, we consider how to compute CTT values based on the *CMK$^+$-tree*.

— CTT computation in the *CMK$^+$-tree(HD)*. Firstly, the search process starts by locating one node in the new hierarchical tree according to a buyer's query. Then, similar to three-dimensional CTT computation (see Section 6.3), three cases should be differentiated in the new hierarchical tree. After that, one or several *CMK-trees* are selected for answering the buyer's query on the remaining three dimensions. The search process in each *CMK-tree* can be found in Algorithm 2.

— CTT computation in the *CMK$^+$-tree(LD)*. The process of three-dimensional CTT computation can be applied to computing CTT values in this case. But the way to compute two VRAs (left border and right border) in each subtree (i.e., extended *MK-tree*) is different, as each VRA is computed based on a two-dimensional plane, rather than a vertical line. For example, in Fig. 15(b), that two-dimensional plane is formed by Transaction Amount dimension and the fourth dimension. Then, each VRA is to compute the number of generated intervals intersecting a two-dimensional plane. Finally, the result still equals to the difference of two VRAs.

6.5.2. High-Dimensional CTT Computation. The construction process of a low-dimensional *CMK$^+$-tree* can be applied to five or higher dimensional CTT computation. However, high-dimensional CTT computation is complex, because the increased dimensions lead to exponential storage space consumption for storing the pre-computed aggregation results. In such a case, the following existing dimension reduction methods can be adopted for high-dimensional CTT computation based on the *CMK$^+$-tree*:

— **Ordering dimensions:** This approach is based on the observation that whether a small number of dimensions bear most of the information. Namely, dimensionality problem can be reduced by ordering dimensions according to their importance so that only information from selected important dimensions is indexed. For example, Lin et al. [1994] propose *TV-tree* to index high-dimensionality data that adopts this idea to avoid the dimensionality problem. Therefore, for high-dimensional CTT computation, we can first order all the transaction dimensions based on their importance, and then maintain the index of low but important dimensionality (e.g., *CMK$^+$-tree*).

— **Mapping dimensions:** This approach is to establish mapping relations between high-dimensional data and low-dimensional data. For example, Ooi et al. [2000] propose to map a point in d-dimension to a 1-D line using the maximum or minimum value of all dimensions. In addition, they also map a d-dimensional query to d subqueries with one query for each dimension. Likewise, Wang et al. [2013] propose a paring function to map d-dimensional points to integers. Following this idea, we can first map linear high-dimensional transaction data to our proposed low-dimensional *CMK$^+$-tree*, based on which to fulfill high-dimensional CTT computation.

7. CMR-TREE^{RS}—THE CMK-TREE WITH REDUCED STORAGE SPACE

In the *CMK-tree*, transaction data and ratings are aggregated at the granularity of days. Though it consumes smaller storage space than the actual data, with significant increase of historical transaction data, the size of the *CMK-tree* will become much larger. In this section, we introduce a new approach *CMK-tree^{RS}*, which reduces storage space consumption for a *CMK-tree*, and offers great benefit to trust management with millions of sellers.

In Section 2.5, we have pointed out that the Hierarchical Temporal Aggregation model with fixed storage space (*HTA^{FS}*) [Zhang et al. 2003] can be applied to CTT computation, in order to control the size of the storage space allocated to a seller for storing aggregation index [Zhang and Wang 2013]. However, as a matter of fact, it is difficult to select the size of the fixed storage space, since the number of distinct products as well as the number of product categories in the transactions traded by different sellers are different. Now let us consider an example. Assume two sellers S_1 and

S_2 have the same volume of transaction data over a period of time. The transactions traded by S_1 are widely distributed in multiple product categories, but seller S_2 has numerous repeated transactions belonging to only a few product categories. For S_1, it is necessary to allocate a relatively large storage space to store aggregation index so as to obtain his/her trustworthiness in each product category and a corresponding sub-category. By contrast, for S_2, it is not necessary to allocate the storage space with the same size as S_1, because most transactions are repeated, leading to less storage space consumption for storing aggregation index.

In practice, the time range in CTT queries is usually *"the latest 1 month"*, *"the latest 3 months"*, *"the latest 6 months"*, or *"the latest 12 months"*. When adopting a *CMK-tree* for CTT computation, the above time ranges can be searched, but the *CMK-tree* consumes a large storage space as the aggregation granularity in the Transaction Time dimension is "days". Alternatively, if we aggregate ratings at a coarse time granularity of months, as depicted in Fig. 12(a), the number of points in a two-dimensional space formed by transactions further decreases, since more repeated transactions exist in a month than on a day. In such a case, the size of storage space consumed for a *CMK-tree* can be reduced to a large extent. However, such a coarse aggregation granularity leads to a serious problem regarding the accuracy of CTT computation. For instance, if the current time is *"August 10"*, to answer a buyer's CTT query with the specified time range as *"the latest 1 month"*, the result can be computed based on either the data of *"August"* only or the data of both *"August"* and *"July"*. However, either way cannot guarantee the accuracy of CTT results. In the worst case, the ratings for computing a CTT value of *"the latest 1 month"* come from one day only or one month plus *"29 days"* (if one month contains *"30 days"*). On average, the CTT result comes from the ratings of 1 month $\pm \frac{1}{2}$month.

In the proposed approach *CMK-tree*^{RS}, a new strategy is adopted, which aggregates ratings with different time granularities for different time periods. In other words, in the *CMK-tree*^{RS}, *the ratings within the latest t days* are aggregated *at a fine time granularity of days*, and *the ratings of t days ago* are aggregated *at a coarse time granularity of weeks*. Taking into account the problem of accuracy of CTT values as mentioned in the above, our work provides a tradeoff between storage space consumption and accuracy. In addition, in practice, t is set to be *"90 days"*, considering the typical time ranges in CTT queries are *"the latest 1 month"*, *"the latest 3 months"*, *"the latest 6 months"* and *"the latest 12 months"*. Therefore, for CTT queries regarding a seller's trustworthiness of *"the latest 1 month"* or *"the latest 3 months"*, there is no accuracy problem in the time dimension. For the queries of *"the latest 6 months"*, on average, ratings included in computation cover 6 months $\pm \frac{1}{2}$week. Likewise, in the case of *"the latest 12 months"*, on average, the ratings used in CTT computation cover 12 months $\pm \frac{1}{2}$week.

7.1. Construction of the CMK-tree^{RS}

This section describes the process of *CMK-tree*^{RS} construction. The algorithm for *CMK-tree*^{RS} construction adds the following operations to Algorithm 1, which is given in Section 6.2 and is used for building a *CMK-tree*:

- If $t_{now} - t_{begin} \leq t + 1$, insert the data of a newly happened transaction into the initial *CMK-tree* based on Algorithm 1. The term t_{begin} denotes the starting time and the term t_{now} denotes the current time. As new transactions happen everyday, in order to guarantee the ratings within the latest t days to be aggregated at a fine time granularity, firstly, it is necessary to perform insertion operations leading to the aggregations of $t + 1$ days. Then, the ratings of t days ago are moved to the aggregation index at a coarse time granularity. Note that this order of operations is necessary as if it is first to remove ratings for the transactions that happened t days ago, the ratings to be aggregated at a fine time granularity will include $t-1$ days only, affecting the accuracy of CTT values.

• Otherwise, find all the transactions that happened no later than $t_{now}-t$ whose corresponding ratings are aggregated by days. Then, the Hierarchical Temporal Aggregation (*HTA*) operations are performed to form the *CMK-tree*^{RS}. Finally, continue to insert the data of a newly happened transaction into the new *CMK-tree*^{RS} based on Algorithm 1.

Next, we explain *HTA* operations in detail. In general, *HTA* operations aim to split one *MK-tree* in the initial *CMK-tree* into two *MK-trees* with ratings to be aggregated at different time granularities. To facilitate discussion, we term the two generated *MK-trees* as *MK-tree*^{day} and *MK-tree*^{week}, respectively. Suppose that the number of transactions traded by a seller at the time point t_{now} belongs to m brand-based product categories, namely, the initial *CMK-tree* at the time point t_{now} includes m *MK-trees*. In the meantime, there are m' ($m' \leq m$) brand-based product categories which contain the transactions with the transaction time no later than $t_{now}-t$, and their corresponding ratings are aggregated with the time granularity of days. Then, the *HTA* operations are performed in the above m' *MK-trees*. For the sake of simplicity, we focus on introducing the following *HTA* operations within one of m' *MK-trees* contained in the initial *CMK-tree*.

— Remove the records maintained in an *MK-tree*, which can index the transactions that happened t days ago. Note that our work also guarantees the space utilization of each node in the *MK-tree* during performing removal operation. For instance, as shown in Fig. 14(e), if the records $<[0,13],[1,3),0,Q_1>$, $<[13,30],[1,3),0,Q_2>$ and $<[30,50],[1,3),0,Q_3>$ in the I-node $In(L)_1$ are removed, the remaining records in $In(L)_1$ as well as the record in $In(L)_2$ are merged. In the meantime, the records in the I-node $In(I)_3$ are updated accordingly. In addition, in a more complex case, if the removed transactions with the transaction time falling into the time range of some records, a new *aB*$^{+}$*-tree* is generated[8]. For example, to remove the transactions that occurred before the time point 2 in Fig. 14(b), briefly speaking, we need to (1) delete the corresponding records in both Ln_1 and Ln_2, (2) merge the remaining records in these two nodes, (3) generate a new *aB*$^{+}$*-tree*, and (4) update the records in $In(L)_1$.

— Standardize the data of transactions that happened t days ago. A standardization operation is to set the transactions occurred in the same week with the same x-coordinate. As stated before, the standardization operation essentially leads to a smaller size *CMK-tree*^{RS}. In Section 7.2, we will further explain how and why the *CMK-tree*^{RS} can also reduce the time of computing CTT values.

— Generate the second *MK-tree*, namely *MK-tree*^{week}, using the standardized results. Consequently, instead of having only one *MK-tree*, each subtree that is external to the *C-tree* has both an *MK-tree*^{day} and an *MK-tree*^{week} in the new *CMK-tree*^{RS}. Fig. 16 illustrates the general structure of the *CMK-tree*^{RS}. Note that if there already exist two *MK-trees* with aggregation index at different time granularities in an external subtree, the *MK-tree*^{week} is updated by inserting the standardized results.

[8]The purpose of generating a new *aB*$^{+}$*-tree* is given in Section 6.2.2. Each generated *aB*$^{+}$*-tree* is to keep aggregation index of transactions in the same brand-based product category whose generated intervals along the Transaction Time dimension intersect with the left border of the corresponding rectangle.

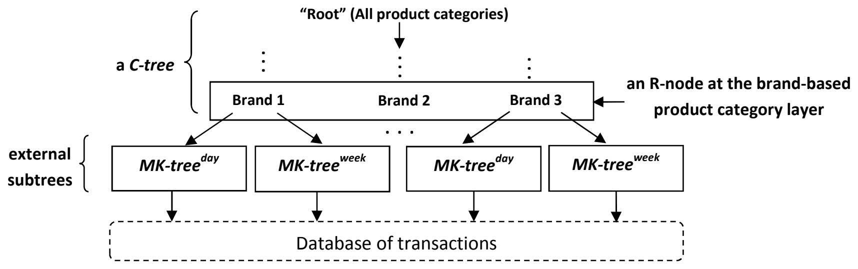


Fig. 16. The structure of the CMK-tree^{RS}

7.2. CTT computation based on the CMK-tree^{RS}

When answering a CTT query based on the new structure *CMK-tree*^{RS}, like Algorithm 2 given in Section 6.3, the *C-tree* is first searched. Then, it computes the left border VRA and the right border VRA in the subtrees that are external to the *C-tree*. Moreover, each external subtree in *CMK-tree*^{RS} has up to two *MK-trees* with aggregated ratings at different time granularities (see Fig. 16). Therefore, the computation of the left border VRA and the right border VRA may also be performed in *MK-tree*^{day} and *MK-tree*^{week} respectively, depending on the query range in the Transaction Time dimension. If the specified time range in a CTT query is $[t_{now} - t, t_{now})$, only the *MK-tree*^{day} is searched; otherwise, the CTT computation algorithm searches both the *MK-tree*^{day} and the *MK-tree*^{week} in the corresponding external subtrees.

Now we introduce an example for further explanation. Suppose that the current time refers to the day of *"July 20"* and t is set to *"90 days"*. For an external subtree in the *CMK-tree*^{RS} that includes two *MK-trees*, the *MK-tree*^{day} maintains the aggregated ratings with the time range [*April 21, July 20*) and the *MK-tree*^{week} maintains the aggregated ratings with the time range [*January 1, April 21*). In this case, if a buyer's CTT query has *"the latest 6 months"* as the time range, the search for computing the left border VRA is performed on the *MK-tree*^{week}. By contrast, for a *CMK-tree* with an *MK-tree* as each external subtree, the *MK-tree* maintains the aggregated ratings for around 200 days in total (i.e., from *January 1* to *July 20*) with the time granularity of days. Though the *CMK-tree*^{RS} has two subtrees: *MK-tree*^{day} and *MK-tree*^{week}, they are much smaller in size. On one hand, the *MK-tree*^{day} includes the aggregates of ratings in a short period of time (i.e., 90 days *vs* 200 days). On the other hand, the *MK-tree*^{week} stores the aggregations for the remaining period of time at a coarse time granularity of "weeks" rather than "days". Therefore, based on the *CMK-tree*^{RS}, the time of computing both the left border VRA and the right border VRA can be reduced. The experiment results to be introduced in Section 8 also have illustrated both storage space reduction and performance improvement of the *CMK-tree*^{RS} in answering buyers' CTT queries.

8. EXPERIMENTS

In this section, we discuss the results of the experiments conducted on four large datasets, which compare the proposed *CMK-tree* and *CMK-tree*^{RS} with three existing approaches *eaR-tree*, *eaP-tree* and *eH-tree* [Zhang et al. 2014] with regards to the aspects of both efficiency in CTT computation and storage space consumption.

Note that the effectiveness of our proposed trust vector based framework has already been studied both analytically and empirically in our earlier work [Zhang et al. 2012b]. In particular, the trust vector based approach can reflect a seller's dynamic trustworthiness in different transaction contexts and identify risks potentially existing in a forthcoming transaction, thus outperforming single-value trust valuation methods [Sabater and Sierra 2001; Wang and Varadharajan 2005b] and a prior trust vector based approach [Wang and Lim 2008].

The *ReputationPro* model proposed in this article focuses on efficient CTT computation for outlining reputation profile as well as the reduction of storage space consumption with new data structures and novel algorithms, i.e., the *CMK-tree* and the *CMK-tree*^{RS}. Our experiments have two parts: while Section 8.3 compares the *CMK-tree* with the existing *eaR-tree*, *eaP-tree* and *eH-tree* in the computation time of CTT values, Section 8.4 further compares the *CMK-tree* and the *CMK-tree*^{RS} in the aspects of both CTT computation time and storage space consumption.

8.1. Datasets

8.1.1. eBay Datasets. With eBay APIs[9], we have obtained detailed feedback and transaction data for up to 90 days of selected 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 within 90 days.

While the products sold by S_1 and S_2 exist in multiple product categories, most products are in the category '*Information, Communication and Media technology*' (see Fig. 8). Specifically, the products sold by S_1 include *MP3 players*, *Notebooks*, *Digital Cameras*, *CD & DVD players*, *LCD monitors* etc, and the products sold by S_2 include *Digital Cameras*, *Video Cameras*, *Camera & Photo Accessories*, *Printers*, *Smartphones* etc. The selection of S_1 and S_2 allows performing both *"drill down"* and *"roll up"* operations in the product category hierarchy (see Fig. 8) when doing finer-grained analysis on a seller's transaction reputation.

8.1.2. Four Large Synthetic Datasets. Considering that only 90 day real transaction data of a seller can be obtained from eBay, and the time range in a CTT query can be *"the latest 6 months"* or *'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 $\mathbf{SD_4(S_2)}$ based on the transaction data of eBay sellers S_1 and S_2. In each synthetic dataset, we expanded the time period of transactions and the daily volume of transactions so as to test the performance of our proposed approaches under the circumstances with exceptionally large volumes of transactions. Specifically, the above four synthetic datasets are further categorized into two types.

(1) **Type I** includes $\mathbf{SD_1(S_1)}$ for S_1 and $\mathbf{SD_3(S_2)}$ for S_2: For each Type I synthetic dataset, we first duplicated the transaction data of each seller 10 times on a given day and thus obtained the transaction data of 10 times as much as the corresponding real dataset. Then, we continued duplicating the newly obtained transactions data of 90 days for about three times for the rest nine months (actually $365 - 90 = 275$ days). E.g., the data of the 91^{st} day duplicates the one of the 1^{st} day. Consequently, with the initial transaction data of 90 days, we obtained the transactions of 12 months. As a result, two datasets contain about $480,000$ and $160,000$ 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: For each Type II synthetic dataset, a transaction was randomly selected from the corresponding sellers eBay real dataset of 90 days. This process was repeated until the size of transaction data on a day within 90 days is 10 times as much as that on the same day in the real dataset. Then we duplicated the data of 90 days for 365 days (12 months). In each Type II synthetic

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

Table III. List of selected products from two popular sellers

seller	selected products
S_1	*Apple iPod nano 16GB (mc696ll/a)* at a price of \$150, *Dell Laptop (Inspiron i17rv-3529dbk)* at a price of \$650, *Canon Powershot Digital Camera (sx40 hs)* at a price of \$380
S_2	*Canon EOS DSLR Camera (T3i)* at a price of \$670, *Kodak Pocket Video Camera (Zi8)* at a price of \$240, *Brother Laser Printer (HL-2220)* at a price of \$90

dataset, basically the proportion of a transaction selling a product on a day or in a month is different to that in the corresponding real dataset.

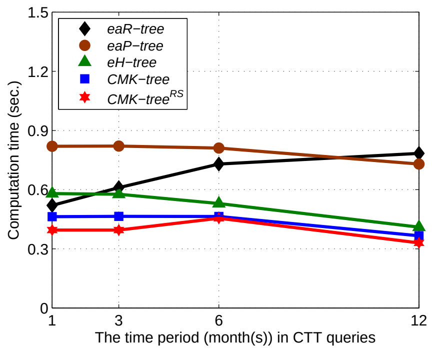
8.2. Experiment Setup

The parameters used in the experiments are as follows: the same page size of 1KB applies to all five index schemes; it is 4 bytes for each of the transaction amount, transaction time, $count\_r$ and sum_r in a record and the *C-value* is of 8 bytes; t in the *CMK-tree*^{RS} is set to be *"90 days"*, i.e., the ratings within the latest *"90 days"* are aggregated at a fine time granularity of days, and the ratings of *"90 days"* ago are aggregated at a coarse time granularity of weeks. In addition, each approach is 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.

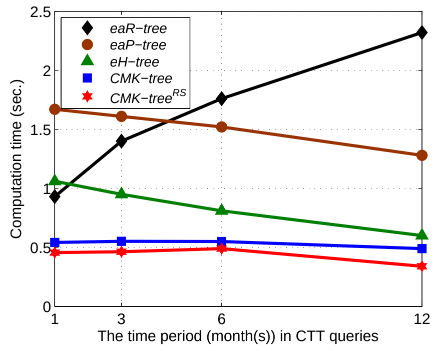
To evaluate the CTT query performance, we measure the CTT values computation time. For each seller, we generated the corresponding queries on either *Transaction Item Specific Trust* (TIST) or *Product Category based Trust* (PCT), covering three transaction dimensions (denoted as **3D CTT queries**), and the queries on *Similar Transaction Amount based Trust* (STAT), covering two transaction dimensions (denoted as **2D CTT queries**).

To generate *3D CTT queries*, based on eBay datasets, we first selected 5 popular products traded by the sellers, each of which has two characteristics: (1) *"roll up"* operations can be preformed continuously at least 3 times along a path in the product category hierarchy; and (2) each product category along the path contains a large number of transactions. Table III lists the selected products for two popular sellers. Then, we set the time range in CTT queries to be "*the latest 1 month*" and computed their TIST values. After that, for each of 5 selected products, *"roll up"* operations were performed continuously 3 times along a path in the product category hierarchy to generate the *3D queries* on PCT. To generate the price range in each PCT query, we adopted a s-trategy to partition the transaction amount range of the current product category that is included in a PCT query. Specifically, as each product category in the product category hierarchy maintains its corresponding transaction amount range $[ta_1, ta_2]$, we partitioned this transaction amount range into 3 equal intervals, and each interval is regarded as an input for the price range in a PCT query. Thus, each product category corresponds to 3 PCT queries with different price ranges. In total, there are 45 $(5 \times 3 \times 3)$ *3D queries* on PCT values. For instance, the generated PCT queries for the product "*Apple iPod nano 16GB (mc696ll/a)*" sold by S_1 (see Table III) are <product-category: "*Apple MP3 player (iPod)*", price-range: "\$100-\$200", time-range: "*the latest 1 month*" $>$ (i.e., PCT at layer 5) and <product-category: "*MP3 player*", price-range: "\$150-\$300", time-range: "*the latest 1 month*" $>$ (i.e., PCT at layer 4). Our experiments also tested the TIST and PCT queries at three different time ranges of *"the latest 3 months"*, *"the latest 6 months"* and *"the latest 12 months"*, respectively. Thus, there are totally 200 (i.e., $(45 + 5) \times 4$) *3D CTT queries* tested for each seller.

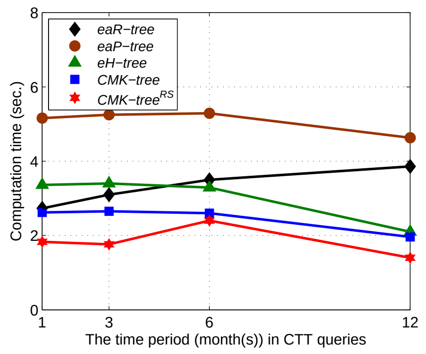
To generate *2D CTT queries*, 45 different price ranges for the above PCT queries are used. In the meantime, the experiments also tested 4 different time ranges of *"the*



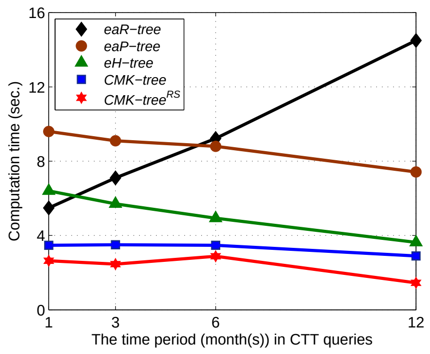
(a) 3D CTT queries on SD1(S1)



(b) 3D CTT queries on SD2(S1)



(c) 2D CTT queries on SD1(S1)



(d) 2D CTT queries on SD2(S1)

Fig. 17. The query performance on two datasets SD1(S1) and SD2(S1) derived from seller S_1

latest 1 month", *"the latest 3 months"*, *"the latest 6 months"* and *"the latest 12 months"*. Thus, there are totally 180 (i.e., 45×4) *2D CTT queries* tested for each seller.

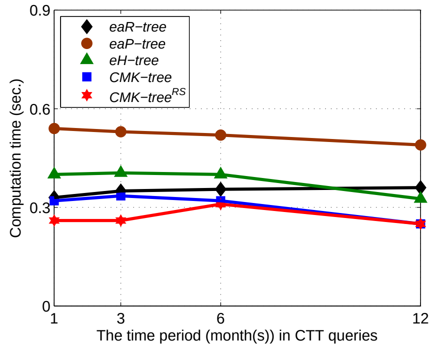
8.3. The Comparison of the CMK-tree and Three Existing Approaches

This section includes the results of the comparison between the *CMK-tree* and three existing approaches *eaR-tree*, *eaP-tree* and *eH-tree* proposed in [Zhang et al. 2014] in terms of CTT values computation time, storage space consumption and the time for tree construction. The experimental results are obtained from the execution on four large synthetic datasets, and the computation time in answering CTT queries is the averaged results of 5 independent runs.

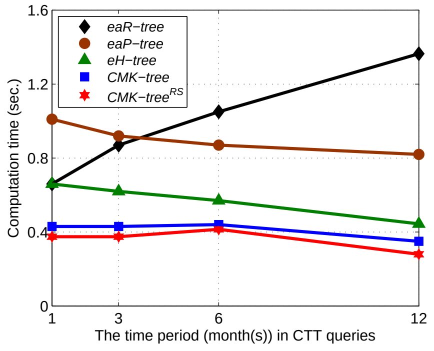
Figure 17 and Figure 18 plot the CTT values computation time of the *eaR-tree*, the *eaP-tree*, the *eH-tree* and the *CMK-tree* in *3D CTT queries* and *2D CTT queries*. First of all, from Figure 17, we can observe the follows.

(1) Compared with other approaches, the performance of the *eaR-tree* shows a different trend in CTT values computation time on both $SD_1(S_1)$ and $SD_2(S_1)$.

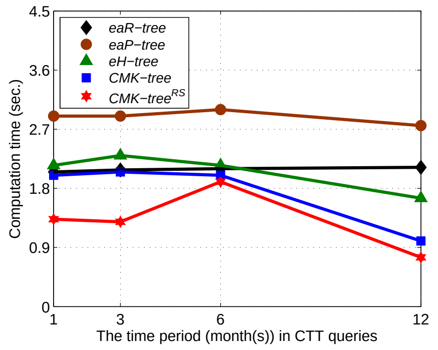
When the time range in a CTT query becomes larger, it means a larger query region for the *eaR-tree*, and the computation time increases almost linearly because more MBRs are overlapped by the expanded query range. In particular, if the time range in CTT queries is *"the latest 12 months"*, the *eaR-tree* has the worst performance in most cases.



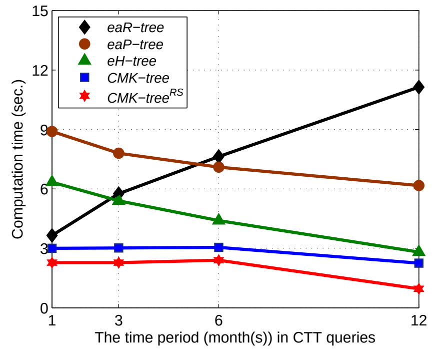
(a) 3D CTT queries on SD3(S2)



(b) 3D CTT queries on SD4(S2)



(c) 2D CTT queries on SD3(S2)



(d) 2D CTT queries on SD4(S2)

Fig. 18. The query performance on two datasets SD3(S2) and SD4(S2) derived from seller S_2

In addition, a similar trend can be observed from the results of *eaP-tree*, *eH-tree* and *CMK-tree*, since they are all based on two VRA queries (see Section 2.4.2) to answer a CTT query. When the time range in a query expands, the computation time is stable or even in decline. In CTT queries, as the time range refers to the latest time period, the right border is always fixed to the time point "*now*". Correspondingly, the time of computing the right border VRA is also fixed for *eaP-tree*, *eH-tree* and *CMK-tree*. However, when the time range in a query expands (i.e., the left border shifts to the left), the time for computing the left border VRA decreases. This is because the above three approaches take advantage of the multi-version structure (see Section 6.1.2), and the versions with their starting time later than the left border of the time range will not be visited. Note that *eaP-tree* and *eH-tree* extend a multi-version *B-tree* (MVBT) [Becker et al. 1996], and *CMK-tree* extends a multi-version *"domain 0" 2-D-B-tree*. As illustrated in Figure 17 and Figure 18, if the time range in a query covers the longest time period, such as *"the latest 12 months"*, most versions of *"domain 0" 2-D-B-trees* (see Fig. 12) have their starting time later than the left border of the time range. In such a case, only a few nodes need to be visited for computing the left border VRA. Consequently, the computation time for each of three approaches *eaP-tree*, *eH-tree* and *CMK-tree* drops to their minimum.

(2) The *CMK-tree* proposed in this article is superior in the efficiency of computing CTT values to the *eaR-tree*, *eaP-tree* and *eH-tree* on both $SD_1(S_1)$ and $SD_2(S_1)$.

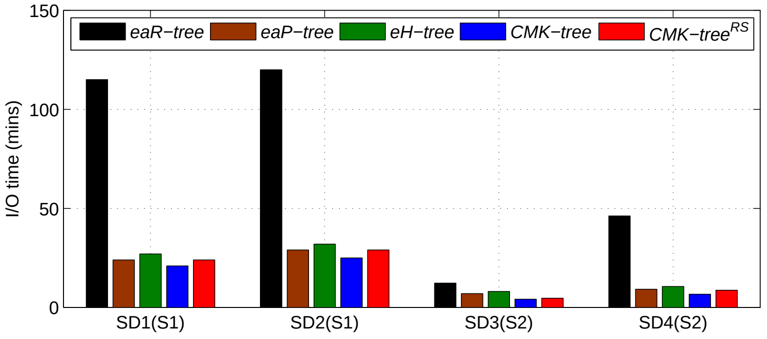


Fig. 19. I/O time for different index schemes construction on four datasets

The *eaR-tree* extends the *aR-tree* [Papadias et al. 2001], and its performance greatly depends on the regions surrounded by the transaction time range and the price range in a CTT query. The *eaP-tree*, which extends the *aP-tree* [Tao et al. 2004], indexes all transactions and cannot essentially aggregate repeated transactions, leading to inferior performance. The *eH-tree*, which improves the *eaP-tree*, is faster than the *eaP-tree* in computing CTT values. The reasons are twofold. On one hand, the *eH-tree* adopts the *aP$^+$-tree* to reduce the time for computing the left border VRA. On the other hand, in the *eH-tree*, the search is done in the fully ordered transaction amount space maintained in an additional *aB$^+$-tree* for computing the right border VRA [Zhang et al. 2014]. However, as the *eH-tree* is still based on the *eaP-tree*, it does not fundamentally resolve the problem existing in the *eaP-tree*. Moreover, the *eH-tree* takes longer time in constructing the aggregation index and consumes more storage space than the *eaP-tree* (see Figure 19 and Figure 20).

By contrast, the *CMK-tree* can not only index each specific product traded in a time period, but also aggregate repeated transactions. More importantly, it delivers shorter and almost stable computation time for answering CTT queries. On average, for the 200 *3D CTT queries* based on Type I synthetic dataset $SD_1(S_1)$, the *CMK-tree* reduces computation time by 33.5% of the *eaR-tree*, by 44.8% of the *eaP-tree*, and by 16.2% of the *eaH-tree*; for the 180 *2D CTT queries*, the *CMK-tree* reduces computation time by 25.2% of the *eaR-tree*, by 51.5% of the *eaP-tree*, and by 18.8% of the *eaH-tree*. In addition, the improvement is more obvious on Type II synthetic dataset $SD_4(S_2)$. On average, for the 200 *3D CTT queries*, the *CMK-tree* reduces computation time by 66.7% of the *eaR-tree*, by 64.9% of the *eaP-tree*, and by 37.6% of the *eaH-tree*; for the 180 *2D CTT queries*, the *CMK-tree* reduces computation time by 63.2% of the *eaR-tree*, by 61.8% of the *eaP-tree*, and by 35.4% of the *eaH-tree*.

From Figure 18, which plots the results executed on $SD_3(S_2)$ and $SD_4(S_2)$ for S_2, we can draw the same conclusion as the one from the results on datasets $SD_1(S_1)$ and $SD_2(S_1)$ for S_1. On average, for the 200 *3D CTT queries* based on Type I synthetic dataset $SD_3(S_2)$, the *CMK-tree* reduces computation time by 12.2% of the *eaR-tree*, by 41.1% of the *eaP-tree*, and by 20.0% of the *eaH-tree*; for the 180 *2D CTT queries*, the *CMK-tree* reduces computation time by 15.6% of the *eaR-tree*, by 39.0% of the *eaP-tree*, and by 17.1% of the *eaH-tree*. On average, for the 200 *3D CTT queries* based on Type II synthetic dataset $SD_4(S_2)$, the *CMK-tree* reduces computation time by 58.0% of the *eaR-tree*, by 54.4% of the *eaP-tree*, and by 28.1% of the *eaH-tree*; for the 180 *2D CTT queries*, the *CMK-tree* reduces computation time by 59.8% of the *eaR-tree*, by 62.2% of the *eaP-tree*, and by 40.3% of the *eaH-tree*.

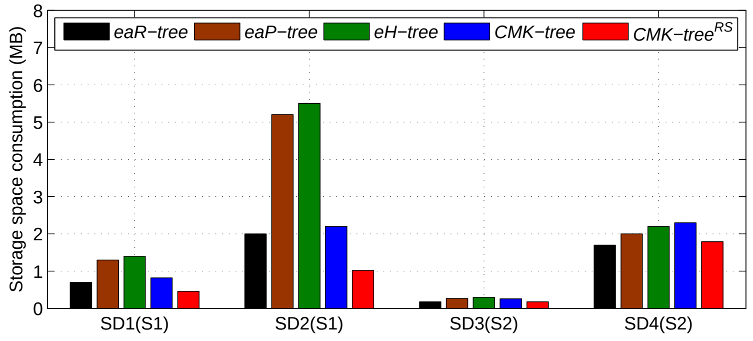


Fig. 20. The storage space consumption for different index schemes on four datasets

Table IV. The comparison of constructing time between *CMK-tree* and the existing index schemes on four datasets

	$SD1(S1)$	$SD2(S1)$	$SD3(S2)$	$SD4(S2)$
CMK-tree vs. *eaR-tree*	18 : 100	21 : 100	34 : 100	16 : 100
CMK-tree vs. *eaP-tree*	88 : 100	86 : 100	60 : 100	73 : 100
CMK-tree vs. *eH-tree*	78 : 100	78 : 100	52 : 100	63 : 100

Table V. The comparison of storage space consumption between *CMK-tree* and the existing index schemes on four datasets

	$SD1(S1)$	$SD2(S1)$	$SD3(S2)$	$SD4(S2)$
CMK-tree vs. *eaR-tree*	117 : 100	110 : 100	144 : 100	135 : 100
CMK-tree vs. *eaP-tree*	63 : 100	42 : 100	96 : 100	115 : 100
CMK-tree vs. *eH-tree*	59 : 100	40 : 100	87 : 100	105 : 100

We also have tested the storage space consumption and I/O time for constructing these aggregation indexes on four synthetic datasets. As plotted in Fig. 19, the *CMK-tree* takes shorter time in construction than three existing approaches on all four datasets (see Table IV for detailed percentage). Overall, the proposed *CMK-tree* leads to 12%-84% time reduction in index construction. However, as plotted in Fig. 20, the *CMK-tree* consumes much storage space in some cases (6 out of 12 cases in Table V) even if it aggregates repeated transactions (see Table V for detailed percentage). For example, on $SD_4(S_2)$ dataset, the *CMK-tree* increases 5%-35% in storage space consumption. This is because, in order to achieve nearly linear query performance, the *CMK-tree* continuously records the transactions whose generated intervals along the Transaction Time dimension intersect with the left border of a rectangle using multiple *aB^+-trees*. Note that each rectangle is formed by a version of *"domain 0" 2-D-B-tree* as shown in Fig. 12. In the next subsection, we will introduce the experimental results delivered by our proposed *CMK-tree*^{RS}, which not only significantly reduces storage space consumption, but also improves CTT computation time.

8.4. The Comparison of the CMK-tree and the CMK-tree^{RS}

Storage space reduction: The Table VI lists the percentage of storage space reduction of the *CMK-tree*^{RS} compared to the *CMK-tree*. Overall, the *CMK-tree*^{RS} reduces 23%-53% in storage space consumption on four synthetic datasets. On average, the reduction is about 38%.

Table VI. *CMK-tree vs. CMK-tree*^{RS}

database	percentage of storage space reduction	accuracy of CTT values
$SD1(S1)$	44%	minimal difference: 0 ; maximal difference: 0.0015 error rate: 1.6%
$SD2(S1)$	53%	minimal difference: 0 ; maximal difference: 0.001 error rate: 1%
$SD3(S2)$	31%	minimal difference: 0 ; maximal difference: 0.028 error rate: 4.2%
$SD4(S2)$	23%	minimal difference: 0 ; maximal difference: 0.0034 error rate: 3.2%

Computation time improvement: the computation time of the two approaches are plotted in Figure 17 and Figure 18. In all cases, the *CMK-tree*^{RS} is faster than the *CMK-tree* in computing CTT values. On average, for the 200 *3D CTT queries* based on four datasets $SD_1(S_1)$, $SD_2(S_1)$, $SD_3(S_2)$ and $SD_4(S_2)$, the *CMK-tree*^{RS} reduces computation time by 10.4%, 18.0%, 11.8% and 12.4% of the *CMK-tree*, respectively; for the 180 *2D CTT queries*, the *CMK-tree*^{RS} reduces computation time by 25.0%, 29.3%, 24.1% and 30.0% of the *CMK-tree*, respectively.

As explained in Section 7.2, compared with the *CMK-tree*, the search in the *CMK-tree*^{RS} for computing the left border VRA and the right border VRA is performed in two subtrees, i.e., *MK-tree*^{day} and *MK-tree*^{week}, but they are much smaller in size than the original *MK-tree* maintained in the *CMK-tree*. Thus, the search time can be reduced in computing two VRA queries. Note that if the time range in a CTT is *"the latest 1 month"* or *"the latest 3 months"*, only the *MK-tree*^{day} is searched for computing both the left border VRA and the right border VRA. Consequently, the *CMK-tree*^{RS} also delivers almost stable computation time for answering CTT queries at the above two time ranges. When the time range in CTT queries is *"the latest 6 months"*, the *MK-tree*^{day} is still searched for computing the right border VRA, but the *MK-tree*^{week} is searched for computing the left border VRA leading to a longer computation time[10]; therefore, we can observe that the computation time increases in this case for the *CMK-tree*^{RS}. However, if the time range in CTT queries is *"the latest 12 months"*, like the *eaP-tree*, the *eH-tree* and the *CMK-tree*, the computation time delivered by the *CMK-tree*^{RS} also drops to the minimum, since only a few nodes are visited for computing the left border VRA.

Accuracy of CTT values: In the meantime, as we have pointed out at the beginning of Section 7, the *CMK-tree*^{RS} reduces the storage space consumption at the expense of the accuracy of CTT values, which exists in the results of the CTT queries regarding *"the latest 6 months"* or *"the latest 12 months"*. Thus, we examined the differences of the results delivered by the *CMK-tree* and the *CMK-tree*^{RS} of the 95 CTT queries (i.e., 50 *3D CTT queries* and 45 *2D CTT queries*) regarding the above two time ranges, i.e., a totally 190 CTT queries are examined. Specifically, in Table VI, we listed the maximal and minimal differences as well as the error rate for the 190 CTT queries on four datasets. Overall, the *CMK-tree*^{RS} leads to 0.001-0.028 as the maximal difference[11] in CTT values in $[0, 1]$ and 2.5% as the average error rate. Therefore, we conclude that the

[10]The *MK-tree*^{day} maintains the aggregated ratings of the latest 3 months, and the *MK-tree*^{week} maintains the aggregated ratings of the remaining 9 months. Although 9 month transaction data and ratings are aggregated at a coarse time granularity of weeks in the *MK-tree*^{week}, it still has a larger size than *MK-tree*^{day}. Thus, the search in the *MK-tree*^{week} consumes more time than that in the *MK-tree*^{day}.

[11]It can be expected that the maximal difference exists in computing the value of *3D CTT queries* with the parameter of product category at a low layer in the product category hierarchy, since a small amount

CMK-tree^{RS} brings a little loss to the accuracy of CTT computation with much gain in storage space reduction and computation time improvement.

9. CONCLUSION AND FUTURE WORK

In this paper, we have addressed the contextual transaction trust problem and proposed a novel and efficient model *ReputationPro* to compute the reputation profile of a seller. It aims to identify the value imbalance problem and context imbalance problem in forthcoming transactions that can cause huge monetary losses to victims. In the meantime, it provides more comprehensive and detailed descriptors for the trustworthiness of a seller. Our proposed index scheme *CMK-tree* takes into account both the static Product Category and dynamic Transaction Amount and Transaction Time dimensions, and achieves nearly linear query performance in answering a buyer's CTT query. This is particularly significant to large-scale transaction data processing. In addition, our proposed *CMK-tree*^{RS} can further reduce the storage space allocated to each seller as well as the time of computing the CTT values with a little loss in accuracy of CTT computation.

For future work, we will focus on improving the performance of the *CMK-tree* with deletion operations. Given a transaction, its generated interval can intersect a number of rectangles until the time point "*now*". Thus, when removing a transaction in a *CMK-tree*, a number of nodes that represent the corresponding rectangles have to be accessed, leading to high complexity. In fact, in the *CMK-tree*, multiple aB^+*-trees* are built to maintain the intersections between the generated intervals and the corresponding rectangles. Therefore, adding indexes to manage these aB^+*-trees* separately would improve the performance of deletion operations. Another direction would be the study of bit-wise machine model based approaches for CTT computation, targeting more efficient solutions. Finally, we will also focus on studying efficient high-dimensional CTT computation approaches.

10. ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable comments which help improve the quality of this paper.

REFERENCES

D. J. Abadi, S. R. Madden, and N. Hachem. 2008. Column-Stores vs. Row-Stores: How Different Are They Really. In *Proceedings of ACM SIGMOD*. 967–980.

S. Adali, R. Escriva, M.K. Goldberg, M. Hayvanovych, M. Magdon-Ismail, B.K. Szymanski, W.A. Wallace, and G. Williams. 2010. Measuring behavioral trust in social networks. In *IEEE International Conference on Intelligence and Security Informatics*. 150–152.

Pankaj K. Agarwala, Lars Argeb, Sathish Govindarajanc, Jun Yanga, and Ke Yi. 2012. Efficient external memory structures for range-aggregate queries. *Computational Geometry* 46, 3 (2012), 358–370.

S. Ba and P. A. Pavlou. 2002. Evidence of the effect of trust building technology in electronic markets: Price premiums and buyer behavior. *MIS Quarterly* 26, 3 (2002), 243–268.

R. Bayer and E. M. McCreight. 1972. Organization and Maintenance of Large Ordered Indices. *Acta Informatica* 1 (1972), 173–189.

B. Becker, S. Gschwind, T. Ohler, B. Seeger, and P. Widmayer. 1996. An asymptotically optimal multiversion B-tree. *The International Journal on Very Large Data Bases* 5, 4 (1996), 264–275.

C. Böhm, S. Berchtold, and D. A. Keim. 2001. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *Comput. Surveys* 33, 3 (2001), 322–373.

A. Caballero, J. Botia, and A. Gomez-Skarmeta. 2007. On the Behaviour of the TRSIM Model for Trust and Reputation. In *German Conference on Multi-Agent System Technologies*. 13–24.

---

of ratings involve in the computation. When performing *"roll up"* operations, more transaction data and ratings are involved in computation. As a result, the computation error rate can be reduced.

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

E. Damiani, S.D.C. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante. 2002. A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks. In *Proceedings of ACM conference on Computer and communications security*. 207–216.

E. Damiani, S.D.C. di Vimercati, P. Samarati, and M. Viviani. 2006. A Wowa-Based Aggregation Technique on Trust Values Connected to Metadata. *Electronic Notes in Theoretical Computer Science* 157, 3 (2006), 131–142.

C. Dellarocas. 2002. Goodwill Hunting: An Economically Efficient Online Feedback Mechanism for Environments with Variable Product Quality. In *Workshop on Agent-Mediated Electronic Commerce*. 238–252.

E. ElSalamouny, V. Sassone, and M. Nielsen. 2010. HMM-Based Trust Model. In *International Workshop on Formal Aspects in Security and Trust*. Vol. 5983. Springer Berlin Heidelberg, 21–35.

J. Golbeck and J. Hendler. 2006. Inferring Binary Trust Relationships in Web-based Social Networks. *ACM Transactions on Internet Technology* 6, 4 (2006), 497–529.

S. Govindarajan, P. K. Agarwal, and L. Arge. 2003. CRB-Tree: An Efficient Indexing Scheme for Range-Aggregate Queries. In *International Conference on Database Theory*. 143–157.

J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, and M. Venkatrao. 1996. Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-Tab, and Subtotals. In *Proceedings of International Conference on Data Engineering*. 152–159.

N. Griffiths. 2005. Task Delegation Using Experience-based Multi-dimensional Trust. In *Proceedings of ACM International Conference on Autonomous Agents and Multiagent Systems*. 489–496.

S. M. Habib, S. Ries, and M. Muhlhauser. 2011. Towards a Trust Management System for Cloud Computing. In *International Conference on Trust, Security and Privacy in Computing and Communications*. 933–939.

F. Ham, E. Imana, A. Ondi, R. Ford, W. Allen, and M. Reedy. 2009. Reputation Prediction in Mobile Ad Hoc Networks Using RBF Neural Networks. In *International Conference on Engineering Applications of Neural Networks*, Vol. 43. 485–494.

V. Harinarayan, A. Rajaraman, and J. D. Ullman. 1996. Implementing Data Cubes Efficiently. In *Proceedings of ACM SIGMOD*. 205–216.

K. Hwang and D. Li. 2010. Trusted Cloud Computing with Secure Resources and Data Coloring. *IEEE Internet Computing* 14, 5 (2010), 14–22.

A. Jøsang. 2001. A Logic for Uncertain Probabilities. *Uncertainty, Fuzziness and Knowledge-Based Systems* 9, 3 (2001), 279–212.

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

A. Jøsang and R. Ismail. 2002. The Beta Reputation System. In *Bled Electronic Commerce Conference*.

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

M. Jurgens and H.-J. Lenz. 1998. The Ra*-tree: An improved R-tree with Materialized Data for Supporting Range Queries on OLAP-Data. In *Workshop on Database and Expert Systems Applications*. Vienna, Austria, 186–191.

S. Kamvar, M. Schlosser, and H. Garcia-Molina. 2003. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of International World Wide Web Conference*. 640–651.

S. T. Kang, Y. D. Chung, and M. H. Kim. 2004. An efficient method for temporal aggregation with range-condition attributes. *Information Sciences* 168, 1-4 (2004), 243–265.

R. Kerr and R. Cohen. 2006. Modelling Trust Using Transactional, Numerical Units. In *Proceedings of ACM International Conference on Privacy, Security and Trust*. 21:1–21:11.

D. J. Kim, D. L. Ferrin, and H. R. Rao. 2008. A trust-based consumer decision-making model in electronic commerce: The role of trust, perceived risk, and their antecedents. *Decision Support Systems* 44, 2 (2008), 342–353.

L. Li and Y. Wang. 2010. Context Based Trust Normalization in Service-Oriented Environments. In *Proceedings of IEEE International Conference on Autonomic and Trusted Computing*. 122–138.

K.-i. Lin, H. V. Jagadish, and C. Faloutsos. 1994. The TV-tree – an index structure for high-dimensional data. *The International Journal on Very Large Data Bases* 3 (1994), 517–542.

W.-Y. Lin and I-C. Kuo. 2004. A Genetic Selection Algorithm for OLAP Data Cubes. *Knowledge and Information Systems* 6, 1 (2004), 83–102.

J. Liu and V. Issarny. 2004. Enhanced Reputation Mechanism for Mobile Ad Hoc Networks. In *Proceedings of International Conference on Trust Management*. 48–62.

X. Liu and A. Datta. 2012. Modeling Context Aware Dynamic Trust Using Hidden Markov Model. In *Proceedings of AAAI Conference on Artificial Intelligence*. Toronto, Canada, 1938–1944.

X. Liu, A. Datta, H. Fang, and J. Zhang. 2012. Detecting Imprudence of 'Reliable' Sellers in Online Auction Sites. In *IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. 246–253.

Z. Malik and A. Bouguettaya. 2009. RATEWEB: Reputation Assessment for Trust Establishment among Web services. *The International Journal on Very Large Data Bases* 18, 4 (2009), 885–911.

S. P. Marsh. 1994. *Formalising Trust as a Computational Concept*. Ph.D. Dissertation. University of Stirling.

L. Mui. 2003. *Computational Models of Trust and Reputation: Agents, Evolutionary Games, and Social Networks*. Ph.D. Dissertation. University of MIT.

T. H. Noor, Q. Z. Sheng, S. Zeadally, and J. Yu. 2013. Trust Management of Services in Cloud Environments: Obstacles and Solutions. *Comput. Surveys* 46, 1 (2013), 1–30.

B. C. Ooi, K.-L. Tan, C. Yu, and S. Bressan. 2000. Indexing the Edges – a Simple and Yet Efficient Approach to High-dimensional Indexing. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. 166–174.

D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. 2001. Efficient OLAP operations in spatial data warehouses. In *Proceedings of International Symposium on Spatial and Temporal Databases*. 443–459.

D. Papadias, Y. F. Tao, P. Kalnis, and J. Zhang. 2002. Indexing Spatio-Temporal Data Warehouses. In *Proceedings of International Conference on Data Engineering*. 166–175.

M. Rehak, M. Pechoucek, and J. M. Bradshaw. 2006. Representing Context for Multiagent Trust Modeling. In *Proceedings of IEEE/WIC/ACM International Conference on Intelligent Agent Technology*. 737–746.

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

B. Rietjens. 2006. Trust and Reputation on eBay: Towards a Legal Framework for Feedback Intermediaries. *Information and Communications Technology Law.* 15, 1 (2006), 55–78.

J. Robinson. 1981. The K-D-B-Tree: A Search Structure For Large Multidimensional Dynamic Indexes. In *Proceedings of ACM SIGMOD*. 10–18.

J. Sabater and C. Sierra. 2001. Regret: Reputation in Gregarious Societies. In *ACM AGENTS*. ACM, 194–195.

J. Sabater and C. Sierra. 2005. Review on Computational Trust and Reputation Models. *Artificial Intelligence Review* 24, 1 (2005), 33–60.

W. Sherchan, S. Nepal, and C. Paris. 2013. A Survey of Trust in Social Networks. *Comput. Surveys* 45, 4 (2013), 1–33.

Carles Sierra and John Debenham. 2007. Information-based agency. In *International Joint Conference on Artificial Intelligence*. 1513–1518.

S. Spitz and Y. Tuchelmann. 2009. A Trust Model Considering the Aspects of Time. In *Proceedings of International Conference on Computer and Electrical Engineering*. 550–554.

G. Suryanarayana and R. N. Taylor. 2002. *A Survey of Trust Management and Resource Discovery Technologies in Peer-to-Peer Applications*. Technical Report. University of California, Irvine.

A. Swaminathan, R. G. Cattelan, Y. Wexler, C. V. Mathew, and D. Kirovski. 2010. Relating Reputation and Money in Online Markets. *ACM Transactions on the Web* 4, 4 (2010), 1–31.

Y. Tao and D. Papadias. 2005. Historical Spatio-temporal Aggregation. *ACM Transactions on Information Systems* 23, 1 (2005), 61–102.

Y. Tao, J. Zhang, D. Papadias, and N. Mamoulis. 2004. Range Aggregate Processing in Spatial Databases. *IEEE Transactions on Knowledge and Data Engineering* 16, 12 (2004), 1555–1570.

S. Toivonen, G. Lenzini, and I. Uusitalo. 2006. Context-aware Trust Evaluation Functions for Dynamic Reconfigurable Systems. In *Models of Trust for the Web Workshop*.

M. Uddin, M. Zulkernine, and S. Ahamed. 2008. CAT: A Context-aware Trust Model for Open and Dynamic Systems. In *Proceedings of ACM Symposium on Applied Computing*. 2024–2029.

S.D.C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, G. Psaila, and P. Samarati. 2012. Integrating Trust Management and Access Control in Data-intensive Web Applications. *ACM Transactions on the Web* 6, 2 (2012), 6:1–6:43.

L.-H. Vu, M. Hauswirth, and K. Aberer. 2005. QoS-based Service Selection and Ranking with Trust and Reputation Management. In *Proceedings of IEEE International Conference on Cooperative information system*. 466–483.

J. Wang, J. Lu, Z. Fang, T. Ge, and C. Chen. 2013. PL-Tree: An Efficient Indexing Method for High-Dimensional Data. In *International Symposium on Spatial and Temporal Databases*. 183–200.

X. F. Wang, L. Liu, and J. Su. 2012. RLM: A General Model for Trust Representation and Aggregation. *IEEE Transaction on Service Computing* 5, 1 (2012), 131–143.

Y. Wang and L. Li. 2011. Two-Dimensional Trust Rating Aggregations in Service-Oriented Applications. *IEEE Transactions on Service Computing* 4, 4 (2011), 257–271.

Y. Wang, L. Li, and G. Liu. 2013. Social context-aware trust inference for trust enhancement in social network based recommendations on service providers. *World Wide Web* (2013), 1–26. DOI:http://dx.doi.org/10.1007/s11280-013-0241-5 Preprint.

Y. Wang and E.-P. Lim. 2008. The evaluation of situational transaction trust in e-service environments. In *Proceedings of IEEE International Conference of Engineering and Business Education*. 265–272.

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

Y. Wang, K.-J. Lin, D. S. Wong, and V. Varadharajan. 2009. Trust Management Towards Service-Oriented Applications. *Service Oriented Computing and Applications* 3, 2 (2009), 129–146.

Y. Wang and M. P. Singh. 2007. Formal trust model for multiagent systems.. In *Proceedings International Joint Conference on Artificial Intelligence*. 1551–1556.

Y. Wang and V. Varadharajan. 2005a. Trust^2: Developing Trust in Peer-to-Peer Environments. In *Proceedings of International Conference on Services Computing*. 24–31.

Y. Wang and V. Varadharajan. 2005b. Two-phase Peer Evaluation in P2P E-commerce Environments. In *Proceedings of International Conference on e-Technology, e-Commerce and e-Service*. 654–657.

Y. Wang and J. Vassileva. 2007. *Toward Trust and Reputation Based Web Service Selection: A Survey*. Technical Report. University of Saskatchewan.

L. Xiong and L. Liu. 2003. A Reputation-Based Trust Model for Peer-to-Peer eCommerce Communities. In *IEEE International Conference on E-Commerce*. 275–284.

L. Xiong and L. Liu. 2004. PeerTrust: Supporting Reputation-Based Trust in Peer-to-Peer Communities. *IEEE Transactions on Knowledge and Data Engineering* 16, 7 (2004), 843–857.

J. Yang and J. Widom. 2003. Incremental computation and maintenance of temporal aggregates. *Proceedings of International Conference on Very Large Data Bases* 12, 3 (2003), 262–283.

D. Zhang, D. Gunopulos, V. J. Tsotras, and B. Seeger. 2003. Temporal and Spatio-Temporal Aggregations over Data Streams Using Multiple Time Granularities. *Information System* 28, 1-2 (2003), 61–84.

D. Zhang, A. Markowetz, V. J. Tsotras, D. Gunopulos, and B. Seeger. 2001. Efficient Computation of Temporal Aggregates with Range Predicates. In *Proceedings of ACM International Symposium on Principles of Database Systems*. 237–245.

D. Zhang, A. Markowetz, V. J. Tsotras, D. Gunopulos, and B. Seeger. 2002. Efficient Aggregation Over Objects with Extent. In *Proceedings of ACM International Symposium on Principles of Database Systems*. 121–132.

D. Zhang, A. Markowetz, V. J. Tsotras, D. Gunopulos, and B. Seeger. 2008. On Computing Temporal Aggregates with Range Predicates. *ACM Transactions on Database Systems* 33, 2 (2008), 1–38.

H. Zhang and Y. Wang. 2013. A Novel Model for Contextual Transaction Trust Computation with Fixed Storage Space in E-Commerce and E-Service Environments. In *Proceedings of IEEE International Conference on Services Computing*. 667–674.

H. Zhang, Y. Wang, and X. Zhang. 2011. Transaction Similarity-Based Contextual Trust Evaluation in E-Commerce and E-Service Environments. In *Proceedings of IEEE International Conference on Web Services*. 500–507.

H. Zhang, Y. Wang, and X. Zhang. 2012a. Efficient Contextual Transaction Trust Computation in E-Commerce Environments. In *Proceedings of IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. 318–325.

H. Zhang, Y. Wang, and X. Zhang. 2012b. A Trust Vector Approach to Transaction Context-Aware Trust Evaluation in E-commerce and E-service Environments. In *Proceedings of IEEE International Conference on Service Oriented Computing and Applications*. 1–8.

H. Zhang, Y. Wang, and X. Zhang. 2014. The Approaches to Contextual Transaction Trust Computation in E-Commerce Environments. *Security and Communication Networks* 7, 9 (2014), 1331–1351.

J. Zhang. 2011. A Survey on Trust Management for VANETs. In *IEEE International Conference on Advanced Information Networking and Applications*. 105–112.

C. Zouridaki, B. L. Mark, M. Hejmo, and R. K. Thomas. 2006. Robust Cooperative Trust Establishment for MANETs. In *ACM Workshop on Security of Ad Hoc and Sensor Networks*. 23–34.