A Fast High Utility Itemsets Mining Algorithm

Ying Liu Wei-keng Liao Alok Choudhary

Electrical and Computer Engineering Department, Northwestern University
Evanston, IL, USA 60208

{yingliu, wkliao, choudhar}@ece.northwestern.edu

ABSTRACT

Association rule mining (ARM) identifies frequent itemsets from databases and generates association rules by considering each item in equal value. However, items are actually different in many aspects in a number of real applications, such as retail marketing, network log, etc. The difference between items makes a strong impact on the decision making in these applications. Therefore, traditional ARM cannot meet the demands arising from these applications. By considering the different values of individual items as utilities, utility mining focuses on identifying the itemsets with high utilities. As "downward closure property" doesn't apply to utility mining, the generation of candidate itemsets is the most costly in terms of time and memory space. In this paper, we present a Two-Phase algorithm to efficiently prune down the number of candidates and can precisely obtain the complete set of high utility itemsets. In the first phase, we propose a model that applies the "transaction-weighted downward closure property" on the search space to expedite the identification of candidates. In the second phase, one extra database scan is performed to identify the high utility itemsets. We also parallelize our algorithm on shared memory multi-process architecture using Common Count Partitioned Database (CCPD) strategy. We verify our algorithm by applying it to both synthetic and real databases. It performs very efficiently in terms of speed and memory cost, and shows good scalability on multiple processors, even on large databases that are difficult for existing algorithms to handle.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications – *data mining.*

General Terms

Algorithms, Design

Keywords

utility mining, association rules mining, downward closure property, transaction-weighted utilization

1. INTRODUCTION

Association rules mining (ARM) [1] is one of the most widely used techniques in data mining and knowledge discovery and has tremendous applications in business, science and other domains. For example, in the business, its applications include retail shelf management, inventory predictions, supply chain management, bundling products marketing. The main objective of ARM is to identify frequently occurring patterns of itemsets. It first finds all the itemsets whose co-occurrence frequency are beyond a minimum support threshold, and then generates rules from the frequent itemsets based on a minimum confidence threshold. Traditional ARM model treat all the items in the database equally by only considering if an item is present in a transaction or not.

The frequent itemsets identified by ARM does not reflect the impact of any other factor except frequency of the presence or absence of an item. Frequent itemsets may only contribute a small portion of the overall profit, whereas non-frequent itemsets may contribute a large portion of the profit. In reality, a retail business may be interested in identifying its most valuable customers (customers who contribute a major fraction of the profits to the company). These are the customers, who may buy full priced items, high margin items, or gourmet items, which may be absent from a large number of transactions because most customers do not buy these items. In a traditional frequency oriented ARM, these transactions representing highly profitable customers may be left out. For instance, {*milk, bread*} may be a frequent itemset with *support* 40%, contributing 4% of the total profit, and the corresponding consumers is Group A, whereas {*birthday cake, birthday card*} may be a non-frequent itemset with *support* 8% (assume support threshold is 10%), contributing 8% of the total profit, and the corresponding consumers is Group B. The marketing professionals must be more interested in promoting the sale of {*birthday cake, birthday card*} by designing promotion campaigns or coupons tailored for Group B (valuable customers), although this itemset is missed by ARM. Another example is web log data. A sequence of webpages visited by a user can be defined as a transaction. Since the number of visits to a webpage and the time spent on a particular webpage is different between different users, the total time spent on a page by a user can be viewed as utility. The website designers can catch the interests or behavior patterns of the customers by looking at the utilities of the page combinations and then consider re-organizing the link structure of their website to cater to the preference of users. Frequency is not sufficient to answer questions, such as whether an itemset is highly profitable, or whether an itemset has a strong impact. Utility mining is likely to be useful in a wide range of practical applications.

Recently, to address the limitation of AMR, a *utility mining* model was defined [2]. Intuitively, utility is a measure of how "useful" (i. e. "profitable") an itemset is. The *utility* of an item or itemset is based on *local transaction utility* and *external utility*. The local transaction utility of an item is defined according to the information stored in a transaction, like the quantity of the item sold in the transaction. The external utility of an item is based on information from resources besides transactions, like a profit table. The external utility can be a measure for describing user preferences. The definition of utility of an itemset X, $u(X)$, is the sum of the utilities of X in all the transactions containing X. The goal of utility mining is to identify high utility itemsets which drive a large portion of the total utility. Traditional ARM model assumes that the utility of each item is always 1 and the sales quantity is either 0 or 1, thus it is only a special case of utility mining, where the utility or the sales quantity of each item could be any number. If $u(X)$ is greater than a utility threshold, X is a high utility itemset, otherwise, it is a low utility itemset. Table 1 is an example of a transaction database where the total utility is 400. The number in each transaction in Table 1(a) is the sales volume of each item, and the external utility of each item is listed in Table 1(b). $u(\{B, D\}) = (6\times10+1\times6) + (10\times10+1\times6) = 172$. {B, D} is a high utility itemset if the utility threshold is set at 120.

Table 1. A transaction database and its utility table

(a) Transaction table. Each row is a transaction. The columns represent the number of items in a particular transaction. TID is the transaction identification number

TID \ ITEM	A	B	C	D	E
T_1	0	0	18	0	1
T_2	0	6	0	1	1
T_3	2	0	1	0	1
T_4	1	0	0	1	1
T_5	0	0	4	0	2
T_6	1	1	0	0	0
T_7	0	10	0	1	1
T_8	3	0	25	3	1
T_9	1	1	0	0	0
T_{10}	0	6	2	0	2

(b) The utility table. The right column displays the profit of each item per unit in dollars

ITEM	PROFIT ($)(per unit)
A	3
B	10
C	1
D	6
E	5

To the best of our knowledge, there is no efficient strategy to find all the high utility itemsets. A naïve attempt may be to eliminate the items that contribute a small portion of the total utility. However, a high utility itemset may consist of some low utility items. Another attempt is to adopt the level-wise searching schema that exists in fast AMR algorithms, such as Apriori [1]. The base of these traditional ARM algorithms is the "downward closure property" (anti-monotone property): any subset of a frequent itemset must also be frequent. That is, only the frequent k-itemsets are exploited to generate potential frequent (k+1)-itemsets. This approach is efficient since a great number of item combinations are pruned at each level. However, this property doesn't apply to the *utility mining* model. For example, $u(D) = 36 < 120$, D is a low utility item, but its superset {B, D} is a high utility itemset. Without this property, the number of candidates generated at each level quickly approaches all the combinations of all the items. For 10^5 items, more than 10^9 2-itemsets candidates may be generated. Moreover, to discover a long pattern, the number of candidates is exorbitantly large. The cost of either computation time or memory is intolerable, regardless of what implementation is applied. *The challenge of utility mining is in restricting the size of the candidate set and simplifying the computation for calculating the utility.*

Nowadays, in any real application, the size of the data set easily goes to hundreds of Mbytes or Gbytes. In order to tackle this challenge, we propose a Two-Phase algorithm to efficiently mine high utility itemsets. In Phase I, we define *transaction-weighted utilization* and propose a model — *transaction-weighted utilization mining*. *Transaction-weighted utilization of an itemset X* is estimated by the sum of the transaction utilities of all the transactions containing X. This model maintains a *Transaction-weighted Downward Closure Property*: any subset of a high transaction-weighted utilization itemset must also be high in transaction-weighted utilization. (Please note we use a new term *transaction-weighted utilization* to distinguish it from *utility*. The focus of this paper is not proposing this new term, but to utilize the property of *transaction-weighted utilization* to help solve the difficulties in utility mining.) Thus, only the combinations of high transaction-weighted utilization itemsets are added into the candidate set at each level. Therefore, the size of the candidate set is substantially reduced during the level-wise search. The memory cost as well as the computation cost is also efficiently reduced. Phase I may overestimate some low utility itemsets as high transaction-weighted utilization itemsets since we use the *transaction-weighted utilization mining* model, but it never underestimates any itemsets. In phase II, only one extra database scan is performed to filter out the overestimated itemsets. The savings provided by Phase I may compensate for the cost incurred by the extra scan during Phase II. As shared memory parallel machines are becoming the dominant type of supercomputers in industry, we parallelize our algorithm on shared memory multi-process architecture using Common Count Partitioned Database (CCPD) scheme. We verify our algorithm by applying it to both synthetic and real databases. It not only performs very efficiently in terms of speed and memory cost compared to the best existing utility mining algorithm [2] (to our best knowledge), but also shows good scalability on multiple processors. Our algorithm easily handles very large databases that existing algorithms cannot handle.

The rest of this paper is organized as follows. Section 2 overviews the related work. Section 3 formally describes the utility mining model. In Section 4, we propose the Two-Phase algorithm. Section 5 presents our parallelization scheme. The experimental results are presented in section 6 and we summarize our work in section 7.

2. RELATED WORK

In the past ten years, a number of traditional ARM algorithms and optimizations have been proposed. The common assumption of them is that each item in a database is equal in weight and the sales quantity is 0 or 1. All of these algorithms exploit the "downward closure property" as proposed in Apriori [1] (all subsets of a frequent itemset must be frequent), such as DHP [3], DIC [4], ECLAT [5], FP-growth [6].

Quantitative association rules mining is introduced in [7], which associates an antecedent with an impact on a target numeric variable. A behavior of a subset is interesting if the statistical distribution of the targeted quantitative variable stands out from the rest. A data-driven algorithm, called "Window", is developed. OPUS [8] is an efficient algorithm to discover quantitative associations rules in dense data sets. The focus of these two algorithms (identifying rules where the antecedent strongly impacts the targeting numeric attribute) is different from ours (identifying those valuable item combinations and the implied valuable customers). The discovery from our work can guide the layout of goods in stores, or promotion campaigns to valuable customers.

Researches that assign different weights to items have been proposed. MINWAL [9] mines the weighted association rules in binary transaction databases based on the k-support bound property. An efficient association rules generation method, WAR [10], focuses on the generation of rules from the available frequent itemsets instead of searching for weighted frequent itemsets. WARM [11] proposes a weighted ARM model where itemset weight is defined as the average weight value of the items comprising this itemset. [12] proposes a scheme that uniformly weights all the transactions without considering the differences among the items. These weighted ARM models are special cases of utility mining.

One of the problems in the field of knowledge discovery is of studying good measures of interestingness of discovered patterns. Some interestingness measures have been proposed [19]. Actionability and unexpectedness are two important subjective measures. According to these measures, a pattern is interesting if the user can take some action by knowing this pattern or it is surprising to the user. Concepts are defined in probabilistic terms.

A concept, *itemset share*, is proposed in [13]. It can be regarded as a utility because it reflects the impact of the sales quantities of items on the cost or profit of an itemset. Itemset share is defined as a fraction of some numerical value, such as total quantity of items sold or total profit. Several heuristics have been proposed and their effectiveness is well evaluated.

A utility mining algorithm is proposed in [14], where the concept of "useful" is defined as an itemset that supports a specific objective that people want to achieve. It focuses on mining the top-K high utility closed patterns that directly support a given business objective. Since the "downward closure property" no longer holds, it develops a new pruning strategy based on a weaker but anti-monotonic condition. Although this work is contributed to utility mining, the definition of utility and the goal of this algorithm in his work are different from those in our work.

An alternative formal definition of utility mining and theoretical model was proposed in [2], where the utility is defined as the combination of utility information in each transaction and additional resources. Since this model cannot rely on "downward closure property" to restrict the number of itemsets to be examined, a heuristics is used to predict whether an itemset should be added to the candidate set. However, the prediction usually overestimates, especially at the beginning stages, where the number of candidates approaches the number of all the combinations of items. The examination of all the combinations is impractical, either in computation cost or in memory space cost, whenever the number of items is large or the utility threshold is low. Although this algorithm is not efficient or scalable, it is by far the best to solve this specific problem. Our work is based on this definition and achieves a significant breakthrough of this problem in terms of computational cost, memory cost and accuracy.

3. UTILITY MINING

The goal of utility mining is to discover all the itemsets whose utility values are beyond a user specified threshold in a transaction database. We start with the definition of a set of terms that leads to the formal definition of utility mining problem. The same terms are given in [2].

- $I = \{i_1, i_2, \ldots, i_m\}$ is a set of items.
- $D = \{T_1, T_2, \ldots, T_n\}$ be a transaction database where each transaction $T_i \in D$ is a subset of I.
- $o(i_p, T_q)$, *local transaction utility value*, represents the quantity of item i_p in transaction T_q. For example, $o(A, T_8) = 3$, in Table 1(a).
- $s(i_p)$, *external utility,* is the value associated with item i_p in the Utility Table. This value reflects the importance of an item, which is independent of transactions. For example, in Table 1(b), the external utility of item A, $s(A)$, is 3.
- $u(i_p, T_q)$, *utility*, the quantitative measure of utility for item i_p in transaction T_q, is defined as $o(i_p,T_q)\times s(i_p)$. For example, $u(A, T_8) = 3 \times 3$, in Table 1.
- $u(X, T_q)$, *utility of an itemset X in transaction T_q*, is defined as $\sum_{i_p\in X} u(i_p,T_q)$, where $X = \{i_1, i_2, \ldots, i_k\}$ is a k-itemset, $X \subseteq T_q$ and $1\le k\le m$.
- *u(X), utility of an itemset X*, is defined as $\sum_{T_q\in D\wedge X\subseteq T_q} u(X,T_q)$.

(3.1)

Utility mining is to find all the high utility itemsets. An itemset X is a *high utility itemset* if $u(X) \geq \varepsilon$, where $X \subseteq I$ and ε is the minimum utility threshold, otherwise, it is a *low utility itemset*. For example, in Table 1, $u(A, T_8) = 3\times3 = 9$, $u(\{A, D, E\}, T_8) = u(A, T_8) + u(D, T_8) + u(E, T_8) = 3\times3 + 3\times6 + 1\times5 = 32$, and $u(\{A, D, E\}) = u(\{A, D, E\}, T_4) + u(\{A, D, E\}, T_8) = 14 + 32 = 46$. If $\varepsilon = 120$, $\{A, D, E\}$ is a low utility itemset.

3.1 Computational Model and Complexity

We now examine the computational model proposed in [2]. We refer this algorithm as MEU (Mining using Expected Utility) for

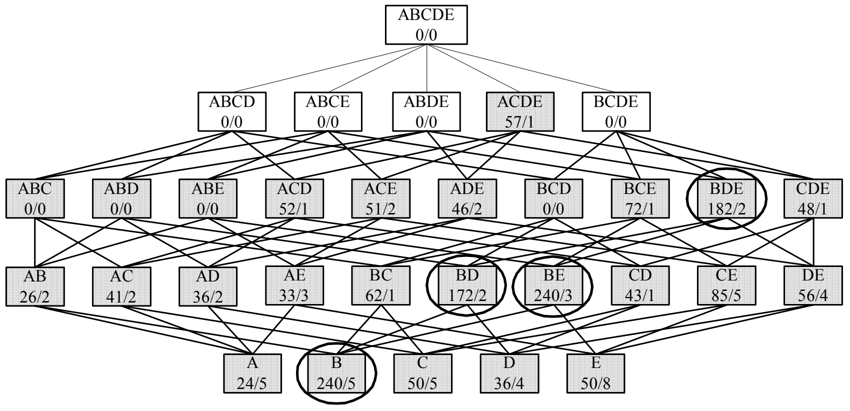


Figure 1. Itemsets lattice related to the example in Table 1. ε = 120. Itemsets in circles are the high utility itemsets. Numbers in each box are utility / number of occurrences. Gray-shaded boxes denote the search space.

the rest of this paper. MEU prunes the search space by predicting the high utility k-itemset, I^k, with the *expected utility value*, denoted as $u'(I^k)$. $u'(I^k)$ is calculated from the utility values of all its (k-1) subsets. If $u'(I^k)$ is greater than ε, I^k is added to the candidate set for k-itemsets, otherwise, I^k is pruned. $u'(I^k)$ is calculated as

$$u'(I^k) = \frac{\sup_{\min}(I^k)}{k-1}\sum_{i=1}^{m}\frac{u(I_i^{k-1})}{\sup(I_i^{k-1})} + \frac{k-m}{k-1}\times\varepsilon \qquad \textbf{(3.2)}$$

I_i^{k-1} is a (k-1)-itemset such that $I_i^{k-1} = I^k - \{i\}$, i.e. I_i^{k-1} includes all the items except item i. sup(I) is the support of itemset I, which is the percentage of all the transactions that contain itemset I. The minimum support among all the (k-1) subsets of I^k is given as

$$\sup_{\min}(I^k) = \min_{\forall I_i^{k-1}\subset I^k,(1\le i\le m)}\{\sup(I_i^{k-1})\} \qquad \textbf{(3.3)}$$

For each I^k, there are k (k-1)-subsets. In (3.2) and (3.3), m is the number of high utility itemsets among the (k-1) subsets where I_i^{k-1} ($1 \le i \le m$) are high utility itemsets, and I_i^{k-1} ($m+1 \le i \le k$) are low utility itemsets. This prediction is based on the *support boundary property* [2] which states that the support of an itemset always decreases as its size increases. Thus, if the support of an itemset is zero, its superset will not appear in the database at all. This approach uses the high utility itemsets at level (k-1) to calculate the expected utility value for level k and the utility threshold ε to substitute the low utility itemsets.

Let us use Table 1 as an example. Figure 1 shows the search space of the database, given ε = 120. Since $u(\{D, E\}) = 56 < \varepsilon$, $\sup_{\min}(\{B, D, E\}) = \min\{\sup(\{B, D\}), \sup(\{B, E\})\} = \min\{0.2, 0.3\} = 0.2$. The expected utility value of {B, D, E} is:

$$\begin{aligned} u'(\{B,C,D\}) &= \frac{0.2}{3-1}\times(\frac{u(\{B,D\})}{\sup(\{B,D\})}+\frac{u(\{B,E\})}{\sup(\{B,E\})})+\frac{3-2}{3-1}\times\varepsilon \\ &= \frac{0.2}{2}\times(\frac{172}{0.2}+\frac{240}{0.3})+\frac{1}{2}\times 120 \\ &= 226 > \varepsilon \end{aligned}$$

Hence, {B, C, D} is a candidate for 3-itemset. Observed from Figure 1, four out of 31 potential candidates are high utility itemsets, which are marked in circles. Using MEU, 26 itemsets (in gray-shaded boxes) have been added into the candidate sets by prediction.

This model somehow reduces the number of candidates; however, it has drawbacks in the following aspects:

1) **Pruning the candidates** – When m is small, the term $\frac{k-m}{k-1}$ is close to 1 or even greater than 1. In this situation, $u'(I^k)$ is most likely greater than ε. When k = 2, $u'(I^2)$ is always greater than ε, no matter m is 0, 1, or 2. Similarly, when k = 3, $u'(I^3) \ge \varepsilon$ if m is 0, 1, or 2. Therefore, this estimation does not prune the candidates effectively at the beginning stages. As shown in Figure 1, all the 2-itemsets and 3-itemsets are included in the candidate sets. If there are 10^5 different items in the database, the number of 2-itemsets is approximately 5×10^9. Such requirements can easily overwhelm the available memory space and computation power of most of the machines.

2) **Accuracy –** This model may miss some high utility itemsets when the variation of the itemset supports is large. For example, if ε = 40 in our example, the expected utility of itemset {C, D, E} is

$$\begin{aligned} u'(\{C,D,E\}) &= \frac{\sup_{\min}(\{C,D,E\})}{3-1}\times(\frac{u(\{C,D\})}{\sup(\{B,C\})}+\frac{u(\{C,E\})}{\sup(\{C,E\})}+\frac{u(\{D,E\})}{\sup(\{D,E\})}) \\ &= \frac{\min\{0.1,0.5,0.4\}}{2}\times(\frac{43}{0.1}+\frac{85}{0.5}+\frac{56}{0.4}) \\ &= 37 < \varepsilon \end{aligned}$$

Therefore, {C, D, E} is predicted to be a low utility itemset and then pruned from the candidate set for 3-itemsets. However, {C, D, E} is indeed a high utility itemset because $u(\{C, D, E\}) = 48 > \varepsilon$.

4. TWO-PHASE ALGORITHM

To address the drawbacks in MEU, we propose a novel Two-Phase algorithm that can highly effectively prune candidate itemsets and simplify the calculation of utility. It substantially reduces the search space and the memory cost and requires less computation. In Phase I, we define a *transaction-weighted* *utilization mining* model that holds a "*Transaction-weighted Downward Closure Property*". (The purpose of introducing this new concept is not to define a new problem, but to utilize its property to prune the search space.) *High transaction-weighted utilization itemsets* are identified in this phase. The size of candidate set is reduced by only considering the supersets of high transaction-weighted utilization itemsets. In Phase II, one database scan is performed to filter out the high transaction-weighted utilization itemsets that are indeed low utility itemsets. This algorithm guarantees that the complete set of high utility itemsets will be identified.

4.1 Phase I

Definition 1. (Transaction Utility) The *transaction utility of transaction* T_q, denoted as $tu(T_q)$, is the sum of the utilities of all items in T_q: $tu(T_q) = \sum_{i_p \in T_q} u(i_p, T_q)$, where $u(i_p, T_q)$ is the same as in Section 3. Table 2 gives the transaction utility for each transaction in Table 1.

Table 2. Transaction utility of the transaction database

TID	Transaction Utility	TID	Transaction Utility
T_1	23	T_6	13
T_2	71	T_7	111
T_3	12	T_8	57
T_4	14	T_9	13
T_5	14	T_{10}	72

Definition 2. (Transaction-weighted Utilization) The *transaction-weighted utilization of an itemset X*, denoted as *twu*(*X*), is the sum of the transaction utilities of all the transactions containing *X*:

$$twu(X) = \sum_{X \subseteq T_q \in D} tu(T_q) \qquad \textbf{(4.1)}$$

For the example in Table 1, $twu(\text{A}) = tu(T_3) + tu(T_4) + tu(T_6) + tu(T_8) + tu(T_9) = 12 + 14 + 13 + 57 + 13 = 109$ and $twu(\{\text{A, D}\}) = tu(T_4) + tu(T_8) = 14 + 57 = 71$.

Definition 3. (High Transaction-weighted Utilization Itemset) For a given itemset *X*, *X* is a *high transaction-weighted utilization itemset* if $twu(X) \geq \varepsilon'$, where ε' is the user specified threshold.

Theorem 1. (Transaction-weighted Downward Closure Property) Let I^k be a *k*-itemset and I^{k-1} be a (*k*-1)-itemset such that $I^{k-1} \subset I^k$. If I^k is a high transaction-weighted utilization itemset, I^{k-1} is a high transaction-weighted utilization itemset.

Proof: Let T_{I^k} be the collection of the transactions containing I^k and $T_{I^{k-1}}$ be the collection of transactions containing I^{k-1}. Since $I^{k-1} \subset I^k$, $T_{I^{k-1}}$ is a superset of T_{I^k}. According to Definition 2,

$$twu(I^{k-1}) = \sum_{I^{k-1} \subseteq T_q \in D} tu(T_q) \geq \sum_{I^k \subseteq T_p \in D} tu(T_p) = twu(I^k) \geq \varepsilon' \qquad \square$$

The *Transaction-weighted Downward Closure Property* indicates that any superset of a low transaction-weighted utilization itemset is low in transaction-weighted utilization. That is, only the combinations of high transaction-weighted utilization (*k*-1)-itemsets could be added into the candidate set C_k at each level.

Theorem 2. Let *HTWU* be the collection of all high transaction-weighted utilization itemsets in a transaction database *D*, and *HU* be the collection of high utility itemsets in *D*. If ε'= ε, then $HU \subseteq HTWU$.

Proof: $\forall X \in HU$, if *X* is a high utility itemset, then

$$\varepsilon' = \varepsilon \leq u(X) = \sum_{X \subseteq T_q} u(X, T_q) = \sum_{X \subseteq T_q} \sum_{i_p \in X} u(i_p, T_q)$$
$$\leq \sum_{X \subseteq T_q} \sum_{i_p \in T_q} u(i_p, T_q) = \sum_{X \subseteq T_q} tu(T_q) = twu(X)$$

Thus, *X* is a high transaction-weighted utilization itemset and $X \in HTWU$. $\square$

According to Theorem 2, we can utilize the *Transaction-weighted Downward Closure Property* in our *transaction-weighted utilization mining* in Phase I by assuming ε' = ε and prune those overestimated itemsets in Phase II.

Figure 2 shows the search space of Phase I. Twelve out of 31 itemsets (in gray-shaded boxes) are generated as candidates (including the single items), and 9 out of 31 itemsets (in circles) are the high transaction-weighted utilization itemsets. The level-wise search stops at the third level, one level less than MEU in Figure 1. (For larger databases, the savings should be more evident.) By holding the *Transaction-weighted Downward Closure Property*, the search space in our algorithm is small. *Transaction-weighted utilization mining* model outperforms MEU in several aspects:

1) **Less candidates** — When ε' is large, the search space can be significantly reduced at the second level and higher levels. As shown in Figure 2, four out of 10 itemsets are pruned because they all contain item A (A is not a high transaction-weighted utilization item). However, in MEU, the prediction hardly prunes any itemset at the beginning stages. In Figure 1, all the 10 2-itemsets are added into the candidate set C_2 because their expected utility values are all greater than ε.

2) **Accuracy** — Based on Theorem 2, if we let ε'=ε, the complete set of high utility itemsets is a subset of the high transaction-weighted utilization itemsets discovered by our *transaction-weighted utilization mining* model. However, the prediction in MEU may miss some high utility itemsets when the variation of itemset supports is large.

3) **Arithmetic complexity** — One of the kernel operations in the Two-Phase algorithm is the calculation for each itemset's transaction-weighted utilization as in equation 4.1. Compared to the calculation for each itemset's expected utility value (equation 3.2) in MEU, equation 4.1 only incurs add operations rather than a number of multiplications. Thus, since the kernel calculation may occur a huge number of times, the overall computation is much less complex.

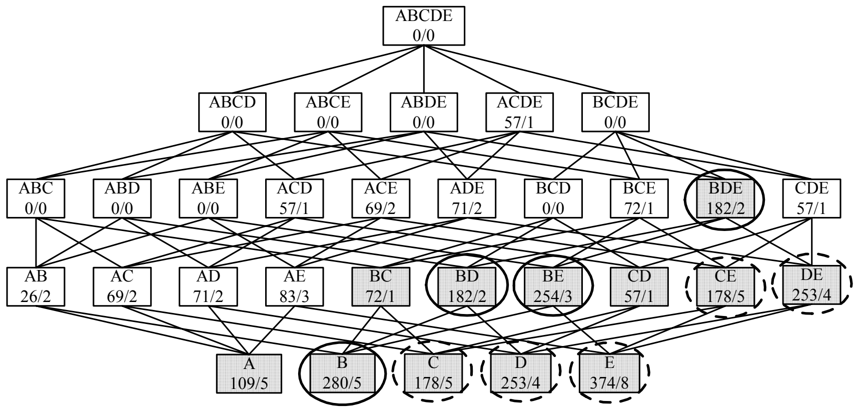


Figure 2. Itemsets lattice related to the example in Table 1. ε' = 120. Itemsets in circles (solid and dashed) are the high transaction-weighted utilization itemsets in *transaction-weighted utilization mining model*. Gray-shaded boxes denote the search space. Itemsets in solid circles are high utility itemsets found by MEU. Numbers in each box are transaction-weighted utilization / number of occurrence.

4.2 Phase II

In Phase II, one database scan is required to select the high utility itemsets from high transaction-weighted utilization itemsets identified in Phase I. The number of the high transaction-weighted utilization itemsets is small when ε' is high. Hence, the time saved in Phase I may compensate for the cost incurred by the extra scan during Phase II. The total computational cost of Phase II is the cost of equation (3.1) × the total number of high transaction-weighted utilization itemsets.

In Figure 2, the high utility itemsets ({B}, {B, D}, {B, E} and {B, D, E}) (in solid black circles) are covered by the high transaction-weighted utilization itemsets (in solid and dashed black circles). Nine itemsets in circles are maintained after Phase I, and one database scan is performed in Phase II to prune 5 of the 9 itemsets since they are not high utility itemsets.

5. PARALLEL IMPLEMENTATION

Since the size of the databases in commercial activities is usually in Gbytes, the computation time or the memory consumption may be intolerable on a single processor. Therefore, high performance parallel computing is highly desired. Due to the fact that shared memory parallel machines are becoming the dominant type of supercomputers in industry because of its simplicity, we design our utility mining parallel implementation on shared-memory architecture.

In shared-memory multi-process architecture (SMPs), each processor has its direct and equal access to all the system's memory as well as its own local caches. To achieve a good scalability on SMPs, each processor must maximize access to local cache and avoid or reduce false sharing. That is, we need to minimize the Ping-Pong effect, where multiple processors might be trying to modify different variables that coincidently reside on the same cache line.

The nature of our Two-Phase utility mining algorithm is a level-wised search method, which is the same as Apriori [1]. [16, 17] has proved that Common Count Partitioned Database (CCPD) is the best strategy to parallel it. In CCPD, data is evenly partitioned on each processor, and each processor traverses its local database partition for incrementing the transaction-weighted utilization of each itemset. All the processors share a single common hash tree, which stores the candidate itemsets at each level of search as well as their transaction-weight utilization. To build the hash tree in parallel, CCPD associated a lock with each leaf node. When a processor wants to insert a candidate into the tree, it starts at root, and successively hashes on the items until it reaches a leaf. It then acquires the lock and inserts the candidate. With this locking mechanism, each processor can insert itemsets in different parts of the hash tree in parallel. For transaction-weight utilization calculation, each processor computes the value from its local database partition.

6. EXPERIMENTAL EVALUATION AND PERFORMANCE STUDY

We evaluate the performance of our Two-Phase algorithm by varying the size of the search space. We also analyze the scalability and result accuracy. All the experiments were performed on a 700-MHz Xeon 8-way shared memory parallel machine with a 4 Gbytes memory, running the Red Hat Linux Advanced Server 2.1 operating system. The program is implemented in C. In parallel implementation, we use OpenMP pragmas [18]. OpenMP is a specification for a set of compiler directives, library routines, and environment variables that can be used to specify shared memory parallelism. Due to its simplicity, OpenMP is quickly becoming one of the most widely used programming styles for SMPs. In SMPs, processors communicate through shared variables in the single memory space. Synchronization is used to coordinate processes. In order to give a fair comparison, we also implement MEU so that both of the implementations can run on the same machine. We use synthetic data and real world data for our evaluation purpose. The details of these databases are described in the following subsections.

6.1 Synthetic Data from IBM Quest Data Generator

We use two sets of synthetic databases from IBM Quest data generator [15]. One is a dense database, T10.I6.DX000K, where the average transaction size is 10; the other is a sparse database,

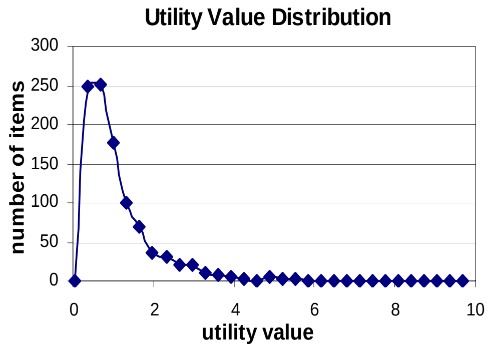


Figure 3. Utility value distribution in utility table.

T20.I6.DX000K, where the average transaction size is 20. The average size of the maximal potentially frequent itemsets is 6 in both sets of databases. In both sets of databases, we vary the number of transactions from 1000K to 8000K, and the number of items from 1K to 8K. However, the IBM Quest data generator only generates the quantity of 0 or 1 for each item in a transaction. In order to fit them into the scenario of utility mining, we randomly generate the quantity of each item in each transaction, ranging from 1 to 5. Utility tables are also synthetically created by assigning a utility value to each item randomly, ranging from 0.01 to 10.00. Observed from real world databases that most items are in the low profit range, we generate the utility values using a log normal distribution. Figure 3 shows the histogram of the utility values of 1000 items.

Table 3. The number of candidate itemsets generated by Phase I of Two-Phase algorithm vs. MEU in the first two database scans

Threshold \ Databases		T10.I6.D1000K		T20.I6.D1000K	
		Phase I	MEU	Phase I	MEU
0.5%	1st *scan*	226128	499500	315615	499500
	2nd *scan*	17	-	18653	-
0.75%	1st *scan*	153181	499500	253116	499500
	2nd *scan*	0	-	1531	-
1%	1st *scan*	98790	499500	203841	499500
	2nd *scan*	0	-	183	-
1.25%	1st *scan*	68265	499500	159330	499500
	2nd *scan*	0	-	33	-
1.5%	1st *scan*	44850	499500	135460	499500
	2nd *scan*	0	-	8	-
1.75%	1st *scan*	27730	499500	104653	499500
	2nd *scan*	0	-	4	-
2%	1st *scan*	16836	499500	84666	499500
	2nd *scan*	0	-	1	-

6.1.1 Number of Candidates

Table 3 presents the number of candidate itemsets generated by Phase I of our Two-Phase algorithm vs. MEU. We only provide the numbers in the first two database scans. The number of items is set at 1000, and the minimum utility threshold varies from 0.5% to 2%. The number of candidate itemsets generated by Phase I at the first database scan decreases dramatically as the threshold goes up. However, the number of candidates generated by MEU is always 499500, which is all the combinations of 1000 items. Phase I generates much fewer candidates compared to MEU. For example, 16836 by Phase I vs. 499500 by MEU at a utility threshold 2% in database T10.I6.D1000K. After the second database scan, the number of candidates generated by our algorithm decreases substantially, mostly decreasing more than 99%. We don't provide the exact numbers for MEU because it actually takes an inordinate amount of time (longer than 10 hours) to complete the second scan. In the case of T20.I6.D1000K, more candidates are generated, because each transaction is longer than that in T10.I6.D1000K. Observed from Table 3, the *Transaction-weighted Downward Closure Property* in transaction-weighted utilization mining model can help prune candidates very effectively.

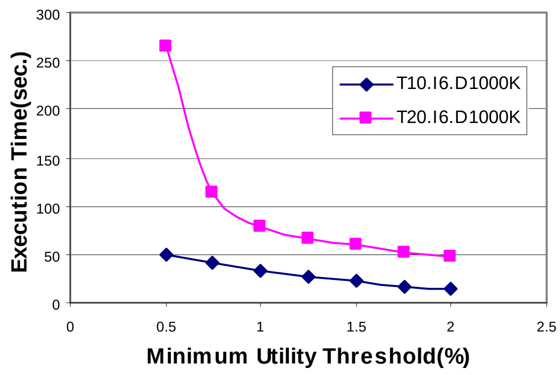


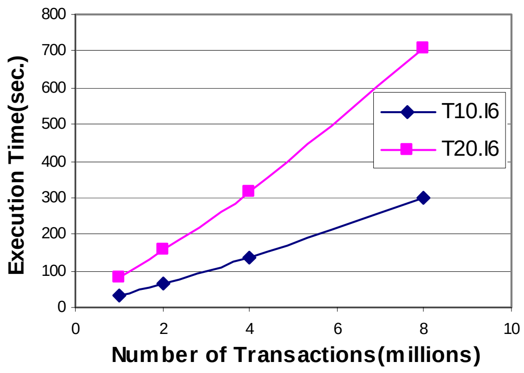
Figure 4. Execution time with varying minimum utility threshold.

6.1.2 Scalability

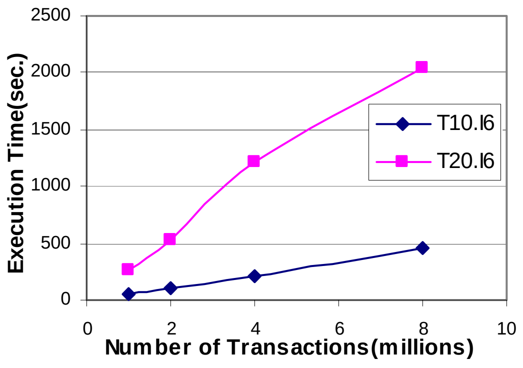
Figure 4 shows the execution time (including both Phase I and Phase II) of the Two-Phase algorithm using T20.I6.D1000K and T10.I6.D1000K. Since the number of candidate itemsets decreases as the minimum utility threshold increases, the execution time decreases, correspondingly. When threshold is 0.5%, the execution time of T20.I6.D1000K is somewhat longer, because it takes 3 more scans over the database in this case compared to other cases with higher threshold values.

Figure 5 presents the scalability of the Two-Phase algorithm by increasing the number of transactions in the database. The number of transactions varies from 1000K to 8000K. The minimum utility threshold is set at 1% to 0.5% in Figure 5(a) and Figure 5(b), respectively. The execution times for either database increase approximately linearly as the data size increases. The execution times for T20.I6.DX000K are longer than that of T10.I6.DX000K, because more computation is required for longer transactions. In addition, the size of database T20.I6.DX000K is larger and therefore takes a longer time to scan.

Figure 6 presents the performance when varying the numbers of items. The number of items varies from 1K to 8K. The minimum utility threshold is set at 1% and 0.5% in Figure 6(a) and Figure 6(b), respectively. When the threshold is 1% as in Figure 6(a), the time decreases as the number of items increases. However, in Figure 6(b), the time for database T10.I6.D1000K with 2K items

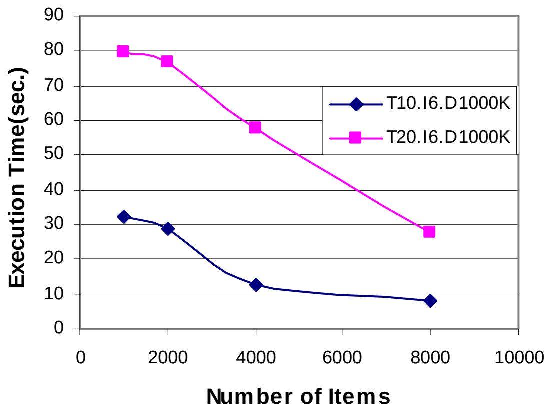


(a) minimum utility threshold = 1%

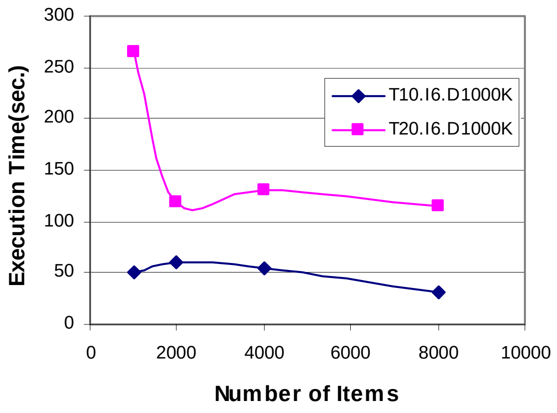


(b) minimum utility threshold = 0.5%

Figure 5. Execution time for databases with different sizes.



(a) minimum utility threshold = 1%



(b) minimum utility threshold = 0.5%

Figure 6. Execution time for databases with different number of items.

is longer than that with 1K items. This is because the total number of candidates is 403651 in the former case, greater than 226145 in the latter case. Similarly, the time for database T20.I6.D1000K with 4K items is longer than that with 2K items, since the total numbers of candidates for the two cases are 1274406 and 779413, respectively. Thus, we can see that the execution time is proportional to the number of candidates generated during the entire process.

6.1.3 Relationship Between Effectiveness vs. Average Transaction Size

As discussed in Section 4 that high transaction-weighted utilization itemsets identified by Phase I (referred as HTWUI in Table 4) cover high utility itemsets (refered as HUI), we would like to investigate the relationship between them. As shown in Table 4, each number of HUI is smaller than the number of HTWUI, correspondingly. Another observation is that the number of HUI is closer to that of HTWUI in database T10.I6.D1000K than in T20.I6.D1000K. This is because in Phase I, the transaction-weighted utilization of any itemset *X* is defined as the sum of the *transaction utilities* of all the transactions containing *X* (equation 4.1). This overestimation gets worse when transactions are longer, because more unrelated items tend to be included in longer transactions. Despite the overestimation, the efficiency of Phase I is still evident. Hence, our proposed algorithm performs more efficiently, especially in dense databases.

Table 4. Comparison of the number of candidate itemsets (CI), high transaction-weighted utilization itemsets (HTWUI), and high utility itemsets (HUI)

Threshold	T10.I6.D1000K			T20.I6.D1000K		
	CI	HTWUI	HUI	CI	HTWUI	HUI
0.5%	226145	711	25	334268	3311	25
0.75%	153181	557	9	254647	1269	9
1%	98790	445	6	204024	799	6
1.25%	68265	370	2	159363	615	2
1.5%	44850	300	0	135468	542	0
1.75%	27730	236	0	104657	465	0
2%	16836	184	0	84667	416	0

6.2 Real-World Market Data

We also evaluated the Two-Phase algorithm using a real world data from a major grocery chain store in California. It contains products from various categories, such as food, health care, gifts, and others. There are 1,112,949 transactions and 46,086 items in the database. Each transaction consists of the products and the sales volume of each product purchased by a customer at a time point. The size of this database is 73MByte. The average transaction length is 7.2. The utility table describes the profit of each product.

In order to evaluate if the utility mining results is useful to the grocery store, we compare the high utility itemsets with the frequent itemsets mined by traditional ARM. We do observe a number of interesting items/itemsets. For example, a kind of bagged fresh vegetable is a frequent item (the support is over 3%), however, its contribution the total profit is less than 0.25%. A combination of two kinds of canned vegetable is also a good example, which occurs in more than 1% of the transactions, but contributes less than 0.25% of the overall profit. Therefore, utility mining can help the marketing professionals in this grocery store make better decisions, such as highlight their highly profitable items/itemsets and reduce the inventory cost for frequent but less profitable items/itemsets.

We evaluate the scalability of our algorithm by varying the threshold. As shown in Table 5, it is fast and scales well. MEU doesn't work with this dataset unless out-of-core technique is designed and implemented, because the number of 2-itemset candidates is so large (approximate 2 billion) that it overwhelms the memory space available to us. Actually, very few machines can afford such a huge memory cost.

Table 5. Experiment summary of the real-world market data

Minimum utility threshold	Running time (seconds)	# Candidates	# High transaction-weighted utilization (Phase I)	# High utility (Phase II)
1%	25.76	11936	9	2
0.75%	33.3	23229	26	3
0.5%	53.09	69425	80	5
0.25%	170.49	627506	457	17
0.1%	1074.94	7332326	3292	80

Result accuracy is a very important feature of utility mining, because the mining results can be used to guide the marketing decisions. Therefore, the accuracy comparison between our Two-Phase algorithm and MEU is given in Table 6. The miss rate is defined as (the number of high utility itemsets – the number of high utility itemsets discovered) ÷ the number of high utility itemsets. To control the execution time of MEU, we set the minimum support and the utility threshold to the same value, i.e. 1%, 0.75%, 0.5%, 0.25% and 0.1%. With this support constraint, MEU works with this data set. However, it may lose some high utility itemsets whose support values are below the support threshold. For example, when the utility threshold is set at 0.1%, the Two-Phase algorithm discovers 80 high utility itemsets whereas MEU (support is set at 0.1%) only gets 66 and misses 14 high utility 2-itemsets. Our algorithm guarantees that all the high utility itemsets will be discovered.

Table 6. Accuracy comparison between Two-Phase algorithm and MEU (with support constraint) on the real-world market data

Threshold	# High utility (Two-Phase)	# High utility (MEU with support constraint)	MEU Miss rate
1%	2	1	50%
0.75%	3	2	33.3%
0.5%	5	3	40%
0.25%	17	17	0%
0.1%	80	66	17.5%

6.3 Parallel Performance

We vary the number of processors from 1 to 8 to study the scalability of our parallel implementation on the real grocery store dataset. Figure 7(a) presents the measured total execution time. The corresponding speedups are presented in Figure 7(b). As the minimum utility threshold decreasing, the search space is increasing dramatically. We observed that it scales better when the searching space increasing. The best case is 4.5 times speedup on 8 processors in the case of minimum threshold = 0.25%. The performance limitation stems from the significant amount of atomic access to the shared hash tree structure. Overall speaking, the parallel scalability in our experiment is good.

7. CONCLUSIONS

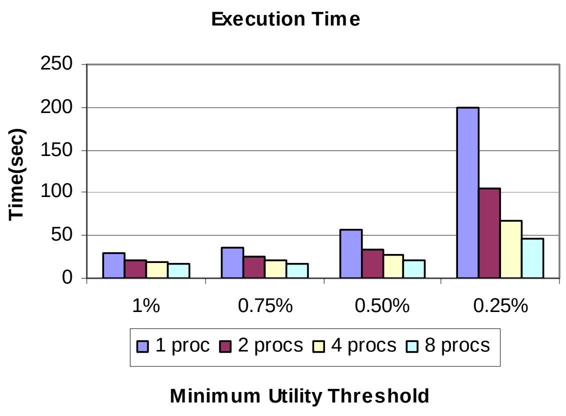
This paper proposed a Two-Phase algorithm that can discover high utility itemsets highly efficiently. Utility mining problem is at the heart of several domains, including retailing business, web log techniques, etc. In Phase I of our algorithm, we defined a term *transaction-weighted utilization,* and proposed the *transaction-weighted utilization mining* model that holds *Transaction-weighted Downward Closure Property*. That is, if a *k*-itemset is a low transaction-weighted utilization itemset, none of its supersets can be a high transaction-weighted utilization itemset. The transaction-weighted utilization mining not only effectively restricts the search space, but also covers all the high utility itemsets. Although Phase I may overestimate some itemsets due to the different definitions, only one extra database scan is needed in Phase II to filter out the overestimated itemsets. Our algorithm requires fewer database scans, less memory space and less computational cost. The accuracy, effectiveness and scalability of the proposed algorithm are demonstrated using both real and synthetic data on shared memory parallel machines. Another important feature is that Two-Phase algorithm can easily handle very large databases for which other existing algorithms are infeasible.

8. ACKNOWLEDGMENTS

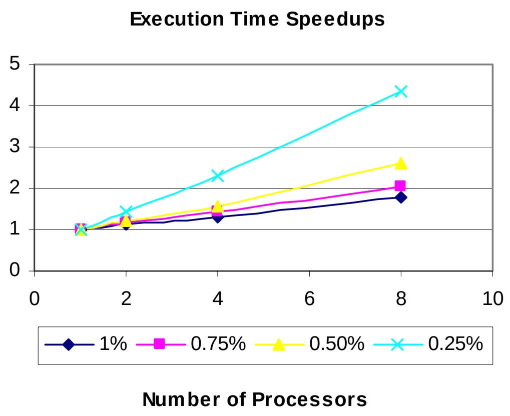
This work was supported in part by NSF grants CCF-0444405, CNS-0406341, CCR-0325207, DOE grant DE-FC02-01ER25485 and Intel Corp.

9. REFERENCES

[1] Agrawal, R., and Srikant, R. Fast algorithms for mining association rules. *20th VLDB Conference*, 1994.



(a) Execution time



(b) Speedups of the execution time

Figure 7. Execution time of the real world database and speedups on Xeon 8-way SMPs. The minimum threshold is varied from 0.25% to 1%.

[2] Yao, H., Hamilton, H. J., and Butz, C. J. A Foundational Approach to Mining Itemset Utilities from Databases. *Proc. of the 4th SIAM International Conference on Data Mining*, Florida, USA, 2004.

[3] Park, J. S., Chen, M., and Yu, P.S. An Effective hash Based Algorithm for Mining Association Rules. *Proc. of ACM SIGMOD Conference*, New York, 1995.

[4] Brin, S., Motwani, R., UIlman J. D., and Tsur, S. Dynamic itemset counting and implication rules for market basket data. *Proc. of ACM SIGMOD Conference on Management of Data*, Arizona, 1997.

[5] Zaki, M. J., Parthasarathy, S., Ogihara, M., and Li, W. New Algorithms for Fast Discovery of Association Rules. *Proc. of 3rd Conference on Knowledge Discovery and Data mining*, California, 1997.

[6] Han, J., Pei, J., and Yin, Y. Mining Frequent Patterns without Candidate Generation. *Proc. of SIGMOD*, 2000.

[7] Aumann, Y., and Lindell, Y. A statistical theory for quantitative association rules. *Proc. of the 5th KDD*, 1999.

[8] Webb, G. I. Discovering associations with numeric variables. *Proc. of the 7th KDD,* 2001.

[9] Cai, C. H., Fu, Ada W. C., Cheng, C. H., and Kwong, W. W. Mining Association Rules with Weighted Items. *Proc. of International Database Engineering and Applications Symposium (IDEAS)*, 1998.

[10] Wang, W., Yang, J., and Yu, P. S. Efficient Mining of Weighted Association Rules (WAR). *Proc. of 6th KDD*, 2000.

[11] Tao, F., Murtagh, F., and Farid, M. Weighted Association Rule Mining using Weighted Support and Significance Framework. *Proc. of International Conference on Knowledge Discovery and Data mining*, 2003.

[12] Lu, S., Hu, H., and Li, F. Mining weighted association rules. *Intelligent Data Analysis*, 5(3) (2001), 211-225.

[13] Barber, B., and Hamilton, H. J. Extracting share frequent itemsets with infrequent subsets. *Data Mining and Knowledge Discovery*, 7(2) (2003), 153-185.

[14] Chan, R., Yang, Q., Shen, Y. Mining high utility Itemsets. *Proc. of IEEE ICDM*, Florida, 2003.

[15] IBM, IBM synthetic data generation code. *http://www.almaden.ibm.com/software/quest/Resources/index.shtml.*

[16] Zaki, M. J., Parallel and Distributed Association Mining: A Survey. *IEEE Concurrency, special issue on Parallel Mechanisms for Data Mining*, Vol. 7, No. 4, December 1999, 4-25.

[17] Zaki, M. J., Ogihara, M., Parthasarathy, S., and Li, W. Parallel Data Mining for Association Rules on Shared-memory Multi-processors. *Supercomputing*, Pittsburg, PA, November 1996.

[18] OpenMP. OpenMP: Simple, Portable, Scalable SMP Programming. *http://www.openmp.org/*.

[19] Silberschatz, A., Tuzhilin, A. What Makes Patterns Interesting in Knowledge Discovery Systems. *IEEE Transactions on Knowledge and Data Engineering,* 8(6), December 1996.