AC-Close: Efficiently Mining Approximate Closed Itemsets by Core Pattern Recovery

Hong Cheng[†]   Philip S. Yu[‡]   Jiawei Han[†]

[†] University of Illinois at Urbana-Champaign
[‡] IBM T. J. Watson Research Center
{hcheng3, hanj}@cs.uiuc.edu, psyu@us.ibm.com

Abstract

Recent studies have proposed methods to discover approximate frequent itemsets in the presence of random noise. By relaxing the rigid requirement of exact frequent pattern mining, some interesting patterns, which would previously be fragmented by exact pattern mining methods due to the random noise or measurement error, are successfully recovered. Unfortunately, a large number of "uninteresting" candidates are explored as well during the mining process, as a result of the relaxed pattern mining methodology. This severely slows down the mining process. Even worse, it is hard for an end user to distinguish the recovered interesting patterns from these uninteresting ones.

In this paper, we propose an efficient algorithm AC-Close *to recover the approximate closed itemsets from "core patterns". By focusing on the so-called core patterns, integrated with a top-down mining and several effective pruning strategies, the algorithm narrows down the search space to those potentially interesting ones. Experimental results show that* AC-Close *substantially outperforms the previously proposed method in terms of efficiency, while delivers a similar set of interesting recovered patterns.*

1. Introduction

Despite the exciting progress in frequent itemset mining [1, 4, 12, 8, 2], an intrinsic problem with the exact mining methods is the rigid definition of support. In real applications, a database contains random noise or measurement error. For example, in a customer transaction database, random noise could be caused by an out-of-stock item, promotions or special events. Measurement error could be caused by noise from experiments, uncertainty involved in discretization. Such random noise can distort the true underlying patterns. In the presence of noise, the exact frequent itemset mining algorithms will discover multiple fragmented patterns, but miss the longer true patterns.

Recent studies [11, 5] try to recover the true patterns in the presence of noise, by allowing a small fraction of errors in the itemsets. While some interesting patterns are recovered, a large number of "uninteresting" candidates are explored during the mining process. Such candidates are uninteresting in the sense that they correspond to no true frequent patterns in the unobserved true transaction database. They are generated as a result of the relaxed mining approach. Let's first examine Example 1 and the definition of the approximate frequent itemset (AFI) [5].

Example 1 Table 1 shows a transaction database D. Let the minimum support threshold $min\_sup = 3$; $\epsilon_r = 1/3$, where ϵ_r is the fraction of noise (0s) allowed in a row (*i.e.*, in a supporting transaction); and $\epsilon_c = 1/2$, where ϵ_c is the fraction of noise allowed in a column (*i.e.*, an item in the set of supporting transactions).

TID	Burger	Coke	Diet Coke
0	1	1	0
1	1	1	0
2	1	0	1
3	1	0	1

Table 1. A Sample Purchase Database D

According to [5], {*burger, coke, diet coke*} is an AFI with support 4, although its exact support is 0 in D. It is deemed uninteresting since *coke* and *diet coke* are seldom purchased together in reality. Those 0s in D are not caused by random noise, but reflect the real data distribution. ∎

To tackle such a problem, we propose to recover the approximate frequent itemsets from "core patterns". Intuitively, an itemset x is a core pattern if its exact support in the noisy database D is no less than αs, where $\alpha \in [0,1]$ is a *core pattern factor*, and s is *min_sup*. Based on the problem formulation, we design an efficient algorithm AC-Close to mine the approximate closed itemsets. A *top-down* mining strategy is exploited, which makes full use of the pruning power of *min_sup* and closeness and thus, narrows down the search space dramatically. Experimental studies show that, AC-Close substantially outperforms AFI in terms of efficiency, while delivers a similar set of the recovered true patterns.

The remainder of the paper is organized as follows. In Section 2, preliminary concepts are introduced. In Section 3, the approximate itemset mining and pruning techniques are introduced. The AC-Close algorithm is presented in Section 4. Experimental results are reported in Section 5 and related work is discussed in Section 6. We conclude our study in Section 7.

2   Preliminary Concepts

Let a transaction database D take the form of an $n \times m$ binary matrix. Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of all items. A subset of I is called an *itemset*. Let T be the set of transactions in D. Each row of D is a transaction $t \in T$ and each column is an item $i \in I$. A transaction t supports an itemset x, if for each item $i \in x$, the corresponding entry $D(t, i) = 1$. An itemset x is *frequent* if the fraction of transactions supporting it is no less than a user-specified threshold *min_sup*.

We use different notations to distinguish the *support* concept of exact frequent itemsets from that of approximate frequent itemsets. The *exact support* of an itemset x is denoted as $sup_e(x)$. The *exact supporting transactions* of x is denoted as $T_e(x)$. For an approximate itemset x, the support is denoted as $sup_a(x)$ and its supporting transaction set is denoted as $T_a(x)$.

Definition 1 *An itemset x is a core pattern in the noisy database D if the exact support of x, $sup_e(x) \geq \alpha s$, where $\alpha \in [0,1]$ is the core pattern factor and s is the min_sup threshold.*

In the presence of random noise, the true frequent patterns would still show some clues, though their supports are affected by the noise. To reflect such a property, we use the core pattern mechanism to recover the true patterns.

In our definition of *approximate frequent itemset*, we adopt from [5] the two variables ϵ_r and ϵ_c as the error thresholds. The definition is given as follows.

Definition 2 *Let ϵ_r, $\epsilon_c \in [0,1]$ and the transaction database be D. An approximate frequent itemset x is (1) a core pattern with $sup_e(x) \geq \alpha s$, and (2) if there exists a set of transactions $T_a(x) \subseteq T$ with $|T_a(x)| \geq s|T|$, s is the min_sup threshold, and the following two conditions hold:*

$$1.\ \forall i \in T_a(x),\ \frac{1}{|x|} \sum_{j \in x} D(i,j) \geq (1 - \epsilon_r)$$

$$2.\ \forall j \in x,\ \frac{1}{|T_a(x)|} \sum_{i \in T_a(x)} D(i,j) \geq (1 - \epsilon_c)$$

ϵ_r and ϵ_c are referred to as row error threshold and column error threshold, respectively.

To further reduce the number of approximate frequent itemsets discovered, we define the concept of approximate closed itemsets.

Definition 3 *An approximate frequent itemset x is closed if there exists no itemset y such that (1) y is a proper superset of x, and (2) $sup_a(y) \geq sup_a(x)$.*

For an approximate pattern x and its superset y, it is possible that $sup_a(y) \geq sup_a(x)$, as caused by the "0-extension" effect [5]. In this case, x is non-closed since y subsumes x in both directions of items and transactions.

The problem of *approximate closed itemset* mining from core patterns is the mining of all itemsets which are (1) core patterns w.r.t. α; (2) approximate frequent itemsets w.r.t. ϵ_r, ϵ_c and *min_sup*; and (3) closed.

3   Approximate Closed Itemset Mining

In this section, we study the problem of mining approximate closed itemsets with ϵ_r, ϵ_c and *min_sup*, and develop several pruning strategies as well.

3.1   Candidate Approximate Itemset Generation

To discover the approximate itemsets, we first mine the set of core patterns with $min\_sup = \alpha s$. These patterns are treated as the initial seeds for possible further extension to approximate frequent itemsets.

Let $\mathcal{C}$ be the set of core patterns. A lattice $\mathcal{L}$ is built over $\mathcal{C}$. Example 2 is used to illustrate the process.

Example 2 Table 2 shows a transaction database D. Let $\epsilon_r = 0.5$, $\epsilon_c = 0.5$, $min\_sup = 3$, and $\alpha = 1/3$.

Mining the core patterns from D with the absolute support $\alpha * min\_sup = 1$ generates 15 core patterns. Figure 1 shows the lattice of core patterns.

TID	a	b	c	d
0	1	1	1	0
1	1	0	0	0
2	1	1	1	1
3	0	0	1	1
4	1	1	0	0
5	1	0	1	1
6	0	1	1	1

Table 2. A Transaction Database D
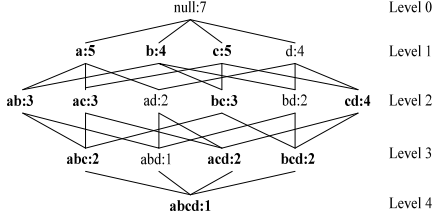
Figure 1. The Lattice $\mathcal{L}$ of Core Patterns

For core pattern $\{a, b, c, d\}$, the number of 0s allowed in a supporting transaction is $\lfloor 4 * 0.5 \rfloor = 2$. Traverse upward in the lattice for 2 levels (*i.e.*, levels 2 and 3), which constitute the *extension space* for $\{a, b, c, d\}$. The extension space is defined as follows.

Definition 4 *For a core pattern* y *with size* l, *the extension space, denoted as* $ES(y)$, *is a set of sub-patterns of* y *from level* $(\lceil l * (1 - \epsilon_r) \rceil)$ *to level* $(l - 1)$ *in the lattice, given* $\epsilon_r \in [0, 1]$.

Because of the ϵ_r fraction of errors allowed in a transaction, for a core itemset y and each sub-pattern $x \in ES(y)$, any transaction supporting x also approximately supports y. According to this property, the transaction set $T_a(y)$ is the *union* of the transaction set $T_e(x)$, $\forall x \in ES(y)$. Thus, for $\{a, b, c, d\}$, its transaction set $T_a(\{a, b, c, d\})$ is the union of the transaction sets of all itemsets at levels 2 and 3 of Figure 1. This property is stated formally in Lemma 1. ∎

Lemma 1 *For a candidate approximate itemset* y *and each pattern* $x \in ES(y)$, *the approximate transaction set* $T_a(y)$ *is computed by taking the union of the exact transaction set* $T_e(x)$.

To summarize, the steps to identify candidate approximate itemsets include: (1) For each core pattern y, identify its extension space $ES(y)$ according to ϵ_r; (2) For each $x \in ES(y)$, take the union of $T_e(x)$; and (3) Check against *min_sup*. Keep those which satisfy *min_sup* as candidate approximate frequent itemsets.

3.2 Pruning by ϵ_c

After we identify the candidate approximate itemsets which satisfy *min_sup* and ϵ_r, the next step is to check each candidate x w.r.t. the column error threshold ϵ_c. According to Definition 2, it needs to scan the approximate transaction set $T_a(x)$. However, a pruning strategy, referred to as ϵ_c *early pruning*, allows us to identify some candidates which violate the ϵ_c constraint without scanning $T_a(x)$. This pruning strategy is stated as follows.

Lemma 2 *Let* $x = i_1 ... i_n$ *be an itemset and the exact support of a single item* $i_k \in x$ *in* D *be* $sup_e(i_k)$, $k \in [1, n]$. *Let the support of the approximate pattern* x *be* $sup_a(x)$. *If*

$$\exists i_k \in x, \frac{sup_e(i_k)}{sup_a(x)} < (1 - \epsilon_c)$$

satisfies, then x *cannot pass the* ϵ_c *check.*

The proof is omitted due to the space limit. The ϵ_c early pruning is effective especially when ϵ_c is small, or there exists an item in x with very low exact support in D. If the pruning condition is not satisfied, a scan on $T_a(x)$ is performed for the ϵ_c check.

3.3 Top-down Mining and Pruning by Closeness

In this section, we show that an efficient *top-down* search strategy can be exploited as the mining framework, thus enabling effective pruning by the closeness definition and the *min_sup* threshold.

A top-down mining starts with the largest pattern in $\mathcal{L}$ and proceeds level by level, in the size decreasing order of core patterns. Let's look at an example.

Example 3 Mining on the lattice $\mathcal{L}$ starts with the largest pattern $\{a, b, c, d\}$. Since the number of 0s allowed in a transaction is 2, its extension space includes patterns at levels 2 and 3. The transaction set $T_a(\{a, b, c, d\})$ is computed by the union operation. Since there exists no approximate itemset which subsumes $\{a, b, c, d\}$, it is an approximate closed itemset.

When the mining proceeds to level 3, for example, the size-3 pattern $\{a, b, c\}$. The number of 0s allowed in a transaction is $\lfloor 3 * 0.5 \rfloor = 1$, so its extension space includes its sub-patterns at level 2.

Since $ES(\{a, b, c\}) \subseteq ES(\{a, b, c, d\})$ holds, $T_a(\{a, b, c\}) \subseteq T_a(\{a, b, c, d\})$ holds too. In this case, after the computation on $\{a, b, c, d\}$, we can prune $\{a, b, c\}$ without actual computation with either of the following two conclusions: (1) if $\{a, b, c, d\}$ satisfies the

min_sup threshold and the ϵ_c constraint, then no matter whether it is closed or non-closed, $\{a, b, c\}$ can be pruned because it will be a non-closed approximate itemset; or (2) if $\{a, b, c, d\}$ does not satisfy the *min_sup*, then $\{a, b, c\}$ can be pruned because it will not satisfy the *min_sup* threshold either. In the first condition, we say *no matter whether $\{a, b, c, d\}$ is closed or non-closed, $\{a, b, c\}$ is non-closed.* This is because, if $\{a, b, c, d\}$ is closed, then $\{a, b, c\}$ is subsumed by $\{a, b, c, d\}$ and thus is non-closed; if $\{a, b, c, d\}$ is non-closed, then there must exist a closed approximate super-pattern which subsumes $\{a, b, c, d\}$, and then subsumes $\{a, b, c\}$ as well. Thus $\{a, b, c\}$ is non-closed. ∎

We refer to the pruning technique in Example 3 as *forward pruning*, which is formally stated in Lemma 3.

Lemma 3 *If $\lfloor (k + 1) \cdot \epsilon_r \rfloor = \lfloor k \cdot \epsilon_r \rfloor + 1$, after the computation on a size-$(k + 1)$ pattern is done, all its size-k sub-patterns in the lattice $\mathcal{L}$ can be pruned with either of the following two conclusions: (1) if the size-$(k + 1)$ pattern satisfies min_sup and ϵ_c, then the size-k patterns can be pruned because they are non-closed; or (2) if the size-$(k + 1)$ pattern does not satisfy min_sup, then the size-k patterns can be pruned because they do not satisfy min_sup either.*

Forward pruning, naturally integrated with the top-down mining, can reduce the search space dramatically due to the *min_sup* and the closeness constraint.

Another pruning strategy, called *backward pruning*, is proposed as well to ensure the closeness constraint, as stated in Lemma 4.

Lemma 4 *If a candidate approximate pattern x satisfies min_sup, ϵ_r and ϵ_c, it has to be checked against each approximate closed itemset y where $x \subseteq y$. If there exists no approximate closed itemset y such that $x \subseteq y$ and $sup_a(x) \leq sup_a(y)$, then x is an approximate closed itemset.*

4 AC-Close Algorithm

Integrating the top-down mining and the various pruning strategies, we proposed an efficient algorithm AC-Close to mine the approximate closed itemsets from the core patterns, presented in Algorithm 1.

5 Experimental Study

A systematic study was performed to evaluate the AC-Close algorithm. We compared it with AFI[1]. In the

---

Input: D, *min_sup* $= s$, ϵ_r, ϵ_c, α
Output: ACI: approximate closed itemsets

1: $\mathcal{C} = $ genFreqItemset(D, αs);
2: $\mathcal{L} = $ buildLattice($\mathcal{C}$);
3: $k = $ max level of $\mathcal{L}$;
4: $C_k = \{x | x \in \mathcal{L} \text{ and } size(x) = k\}$;
5: **repeat**
6:      **for** $x \in C_k$
7:          $ES = $ genExtensionSpace(x, $\mathcal{L}$);
8:          $T_a(x) = $ unionTransaction(ES);
9:          **if** (x not satisfies s or ϵ_c)
10:             $C_k = C_k - \{x\}$;
11:         **if**($\lfloor k \cdot \epsilon_r \rfloor = \lfloor (k - 1) \cdot \epsilon_r \rfloor + 1$)
12:             $C_{k-1} = $ forwardPrune(C_{k-1}, x);
13:     **end for**
14:     $L_k = $ backwardPrune(C_k, ACI);
15:     $ACI = ACI \cup L_k$;
16:     $k = k - 1$;
17: **until** $k = 0$
18: **return** ACI

Algorithm 1: The AC-Close Algorithm

first group of experiments, the efficiency of AFI and AC-Close were tested. In the second group, the result quality of AFI and AC-Close were tested on synthetic datasets with a controlled fraction of random noise embedded and with known underlying patterns.

5.1 Scalability

The IBM synthetic data generator is used to generate datasets for test. A dataset T10.I100.D20K [1] is generated with 20K transactions, 100 distinct items and an average of 10 items per transaction.

Figure 2 (a) shows the running time of both algorithms by varying *min_sup* with $\epsilon_r = \epsilon_c = 0.20$. $\alpha = 0.8$ is used in AC-Close. As shown in the figure, AC-Close runs much faster than AFI.

Figure 2 (b) shows the running time of both algorithms by varying ϵ_r and ϵ_c. To reduce the parameter space, we set $\epsilon = \epsilon_r = \epsilon_c$. *min_sup* $= 0.8\%$ and $\alpha = 0.8$ are used. The running time of AFI increases very quickly as ϵ increases while the efficiency of AC-Close is not affected much by ϵ.

Figure 2 (c) shows the running time of AC-Close by varying the core pattern factor α. $\epsilon_r = \epsilon_c = 0.20$ is used. As α decreases, the core pattern set becomes larger. Therefore, more candidates are generated for approximate itemset mining and the computation time increases. Nevertheless, AC-Close is shown to be very
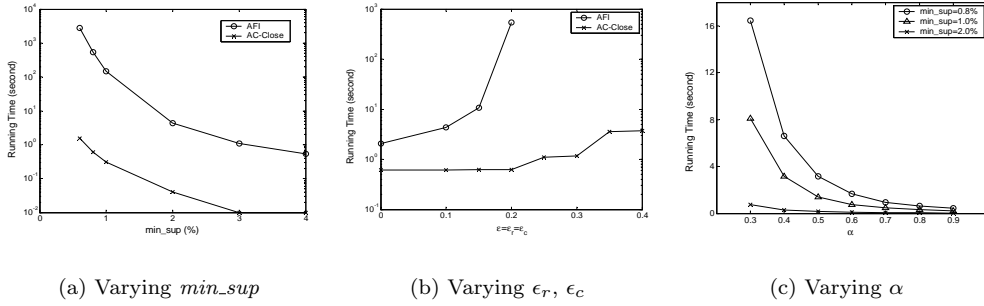
(a) Varying min_sup　　(b) Varying ϵ_r, ϵ_c　　(c) Varying α

Figure 2. Running Time of AFI **and** AC-Close**, on T10.I100.D20K**

(a) Varying Number of Transactions　　(b) Varying Number of Items　　(c) Varying Avg Number of Items Per Transaction
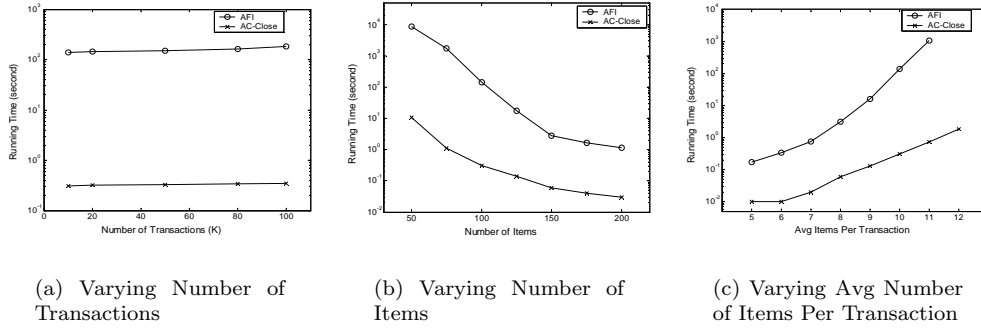
Figure 3. Running Time of AFI **and** AC-Close**, Vary Database Statistics**

efficient even when α is set to as low as 0.3.

Figure 3 shows the scalability tests by varying different statistics of the transaction database. $min\_sup = 1\%$, $\epsilon_r = \epsilon_c = 0.20$ and $\alpha = 0.8$ are used.

Figure 3 (a) shows the running time by varying the number of transactions. The number of distinct items is 100 and the average items per transaction is 10. Since both algorithms only perform the union or intersection operations on the transaction id lists without scanning the database repeatedly, increasing the transaction number does not affect the performance much.

Figure 3 (b) shows the running time by varying the number of distinct items in the database. The number of transactions is 10K and the average transaction length is 10. When the number of items is large, the database becomes sparse. So both algorithms run more efficiently. As the number of items decreases, the number of approximate frequent itemsets increases. Figure 3 (b) shows that the running time of AFI increases more rapidly than that of AC-Close.

Figure 3 (c) shows the running time by varying the average number of items per transaction. The number of transactions is 10K and the number of distinct items is 100. As the average transaction length increases,

the combination between items increases exponentially. The figure shows that the running time of AFI increases much faster.

5.2 Quality

We test and compare the quality of mining results by AFI and AC-Close. A dataset D_t, T10.I100.D20K, is used as the noise-free dataset. Noise is introduced to flip each entry of D_t with a probability p. The noisy dataset is denoted as D. Exact frequent pattern mining is applied to D_t and the results are treated as the ground truth, denoted as F_{true}. In addition, AFI and AC-Close are applied to D and the approximate itemsets are treated as the recovered patterns, denoted as F_{apr}. Two evaluation metrics *precision* and *recall* are used as the measure, as defined below.

$$precision = \frac{|F_{true} \cap F_{apr}|}{|F_{apr}|}, \quad recall = \frac{|F_{true} \cap F_{apr}|}{|F_{true}|}$$

Table 3 shows the quality comparison between AFI and AC-Close with $p = 0.05$. $\epsilon_r = \epsilon_c = 0.20$ and $\alpha = 0.8$ are used. As we can see, AFI achieves the same

recall values as AC-Close but AFI also generates some false positive patterns which do not appear in F_{true}. In contrast, the precision of AC-Close is 100%.

Table 3. Precision and Recall of Mining Result by AFI and AC-Close, Noise Level $p = 0.05$

$min\_sup(\%)$	Precision		Recall	
	AFI	AC-Close	AFI	AC-Close
0.6	90.04	100	76.70	76.70
0.8	90.46	100	78.61	78.61
1.0	91.69	100	78.30	78.30
2.0	97.52	100	81.71	81.71
3.0	99.40	100	81.61	81.61

Table 4. Precision and Recall of the Mining Result by AC-Close, varying α, $p = 0.05$

α	min_sup=0.8%		min_sup=1.0%	
	Precision	Recall	Precision	Recall
0.9	100	78.61	100	78.30
0.7	100	78.61	100	78.30
0.5	100	78.61	100	78.30
0.3	99.81	78.61	99.94	78.30
AFI	90.46	78.61	91.69	78.30

Table 4 shows the result quality of AC-Close at different α values, at a noise level $p = 0.05$. $\epsilon_r = \epsilon_c = 0.20$ is used. At the bottom row of the table, the precision and recall of AFI is shown as the comparison baseline. As shown in Table 4, a large α value achieves the same recall as a small α when the noise level is low.

6 Related Work

Error-tolerant itemset (ETI) mining was first studied by [11]. Two error-tolerant models, weak ETI and strong ETI are proposed. Some heuristics are used to greedily search for ETIs.

Liu et al. [5] proposed an approximate frequent itemset (AFI) model which controls errors of two directions in the data matrices formed by transactions and items.

Other related studies on approximate frequent patterns include [6, 3, 7, 10, 9]. [6] showed that approximate association rules are interesting and useful. [3] proposed the concept of free-sets and led to an error-bound approximation of frequencies. The goal of [7]

is to derive a compact representation – *condensed frequent pattern base*. In the support envelope study [10], a symmetric ETI model was proposed. [9] proposed to mine the dense itemsets – the itemsets with a sufficiently large submatrix that exceeds a given density threshold.

7 Conclusions

In this paper, we propose an effective algorithm AC-Close for discovering approximate closed itemsets from transaction databases in the presence of random noise. Core pattern is proposed as a mechanism for efficiently identifying the potentially true patterns. Experimental study demonstrates that our method can efficiently discover a large fraction of true patterns while avoiding generating false positive patterns.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. VLDB'94*, pages 487–499.

[2] R. J. Bayardo. Efficiently mining long patterns from databases. In *Proc. SIGMOD'98*, pages 85–93.

[3] J. Boulicaut, A. Bykowski, and C. Rigotti. Approximation of frequency queries by means of free-sets. In *Proc. PKDD'00*, pages 75–85.

[4] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. SIGMOD'00*, pages 1–12.

[5] J. Liu, S. Paulsen, X. Sun, W. Wang, A. Nobel, and J. Prins. Mining approximate frequent itemsets in the presence of noise: Algorithm and analysis. In *Proc. SDM'06*, pages 405–416.

[6] H. Mannila and H. Toivonen. Multiple uses of frequent sets and condensed representations. In *Proc. KDD'96*, pages 189–194.

[7] J. Pei, G. Dong, W. Zou, and J. Han. Mining condensed frequent pattern bases. *Knowledge and Information Systems*, 6(5):570–594, 2004.

[8] J. Pei, J. Han, and R. Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *Proc. DMKD'00*, pages 21–30.

[9] J. Seppänen and H. Mannila. Dense itemsets. In *Proc. KDD'04*, pages 683–688.

[10] M. Steinbach, P. Tan, and V. Kumar. Support envelopes: A technique for exploring the structure of association patterns. In *Proc. KDD'04*, pages 296–305.

[11] C. Yang, U. Fayyad, and P. S. Bradley. Efficient discovery of error-tolerant frequent itemsets in high dimensions. In *Proc. KDD'01*, pages 194–203.

[12] M. J. Zaki. Scalable algorithms for association mining. *IEEE Trans. Knowledge and Data Engineering*, 12(2):372–390, 2000.