

# Discovering Fully Dependent Patterns\*

*Feng Liang, Sheng Ma, and Joseph L. Hellerstein<sup>†</sup>*

## 1 Introduction

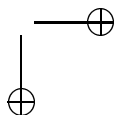
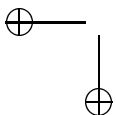
As it becomes feasible to collect large volumes of data, businesses are increasingly looking for ways to capitalize on these data, especially market data. To date, the focus has been frequent patterns, especially frequent association rules. However, in applications such as detecting anomalies in computer networks and identifying security intrusions, there is much more interest in patterns that predict undesirable situations, such as service disruptions. Such patterns are often infrequent (at least in well managed systems) and are characterized by statistical dependency rather than their frequency. Unfortunately, the statistical dependency based on the dependency test yields neither upward nor downward closure, and hence efficient algorithms cannot be constructed. Herein, we circumvent this problem by proposing fully dependent patterns, d-patterns. D-patterns are defined so as to ensure downward closure, which makes it possible for us to construct an efficient algorithm for their discovery. We apply our algorithm to data from a network at a large insurance company and show that several patterns of interest are discovered[8]. For example, a group of hosts generated port-scan events three times in a week. This provides a possible indicator of a security intrusion. In another example, we observed three events: network interface card failure, unreachable destination, and “cold start” trap often occurred together, although not frequent. The last event indicates that the router has failed and restarted. Then, the first two events may provide advance warning of when the third will occur.

Our work is motivated by issues we have encountered in discovering patterns of events in computer networks. First, as indicated above, we are concerned with how to discover infrequent, but dependent itemsets. In computer networks, dependent temporal event sequences provide knowledge to predict later events, which is of particular interest if these event are related to malfunctions and/or service dis-

---

\*F

<sup>†</sup>IBM T. J. Watson Research Center, 30 Sawmill River Road, Hawthorne, NY 10532.



ruptions. Unfortunately, existing mining techniques require the support thresholds to be set very low in order to discover infrequent patterns. This results in a large number of unimportant patterns mixed in with a few patterns of interest.

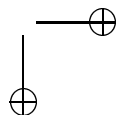
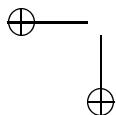
Second, we need to deal with data collected in a noisy environment. In networks, data may be lost because of severed communication lines or router buffer overflows. In help desk systems, data may be corrupted because of human errors. Some valid patterns will be missed at the presence of noise. To illustrate, suppose we have a 15-item pattern with true frequency 15% and the minimal support is set to be 10%. Assume that we get the data through a transmission channel in which each item could be lost with probability 5%. Due to the missing information, the observed frequency will be  $15\% \times (0.95)^{15} \approx 7\%$ , which is less than the minimal support. With a little more calculation, we see that only sub-patterns with length no greater than 7 would satisfy the minimal support 10%. Consequently, instead of reporting one long pattern with length 15, over 6435 subpatterns with length seven or less are found and reported as the maximal frequent itemsets. Clearly, the problem is due to the fixed minimum support threshold that favors short itemsets over long ones.

Third, we are concerned with skewed distributions of items. In our experience with alarms in computer systems, 90% events are often generated by only 10% or 20% hosts. Liu et al.[10] has argued that this is a major obstacle to applying traditional association mining in which a minimum support is fixed. This motivates Liu [10] to use multiple minimum support thresholds that take into account the distribution of items and the length of an itemset. However, this introduces extra parameters, which complicates pattern discovery.

Clearly, the above three issues are directly related to the first step of the association rule discovery, i.e., finding all frequent patterns with a fixed threshold. To address these issues, we employ the hypothesis test, which has long been used to test whether a set of variables significantly exceed an assumed baseline distribution. When there is no specific prior knowledge about the association in a data set, the reasonable baseline distribution is the independent distribution. Any departure from the independent distribution will be of interest. Such a hypothesis test is known as the dependency test.

However, the dependency test are neither downward nor upward closed, a situation that impairs the efficiency of pattern discovery. Although we can apply the dependency test in the post-processing of all frequent itemsets, doing so limits our ability to discover infrequent but dependent itemsets. This motivates us to introduce a **fully dependent** patterns (d-patterns). A d-pattern requires all its subsets to satisfy the dependency test. Clearly, the full dependency is very stringent dependency. We will further discuss this and provide solutions that combine the strength of the frequency test and consider the possible negative dependency.

The dependency test (essentially the chi-square test), has been used previously. For example, Brin et al.[2] used it for testing the dependence of a set of variables. Ma et al.[11] used it for testing the periodicity. However, all these work used the normal approximation that assumes that the sample size is large. In this work, we provide a thorough treatment for mining infrequent, but fully dependent itemsets at the presence of noise. In particular, we analytically and empirically quantify the



effect of the randomness when itemsets occur infrequently in a noisy situation. To our best knowledge, this has not been considered before in the context for pattern discovery. Extensive experiment have been conducted to compare the performance of d-patterns against frequent patterns. We show that d-patterns can discover infrequent patterns in a noisy situation, while the traditional approach fails. We also apply our algorithm to real data. Many infrequent d-patterns are found, which can not be discovered otherwise.

## 1.1 Related work

There have been some interests recently in infrequent patterns. Cohn et al.[5] define a symmetric similarity measurement as the ratio of the support of a itemset divided by the summation of the supports of its items. However, the focus is on two items that are highly correlated.

The problem of skewed item distributions has been addressed by a couple of authors. Brin et al. [2] analyze this problem and introduce a chi-square test. However, their work focuses on the variable level dependency test, while we focus on the item level testing. Liu et al. [10] develop an algorithm with multiple support thresholds. Yang et al. [18] propose an approach based on information gain. None of these efforts address infrequent, noisy itemsets.

Mannila et al. [14] introduce frequent episodes, a generalization of association rules for temporal data. Recently, [12] and [13] characterize temporal events from the statistical perspective. However, once again, there is no consideration for infrequent patterns.

## 1.2 Organization of the paper

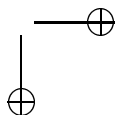
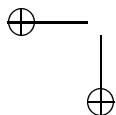
The remainder of this paper is organized as follows. Section 2 motivates, defines and analyzes d-patterns. Section 3 shows some experimental results. Our conclusions are contained in Section 4.

# 2 Problem Formulation

## 2.1 Probabilistic Model for Transaction Data

Let  $\mathcal{I}$  be a set of  $k$  distinct items. A transaction  $T$  is a random variable taking values of all possible subsets of  $\mathcal{I}$ . Assume that there exists an unknown distribution  $P$  on the  $2^k$  possible states of  $T$ . A transaction data  $\mathcal{T} = \{T_1, \dots, T_n : T_i \stackrel{i.i.d}{\sim} P\}$  is a collection of  $n$  independent samples from  $P$ . For example, we could assume that there exists a distribution for people's shopping behavior at a super market and each customer does his/her shopping independently. The market basket data are some independent instances from that unknown distribution.

Given an item set  $E = \{i_1 i_2 \dots i_m\}$ , let  $c(E)$  denote the count of transactions containing  $E$ . To distinguish dependent itemsets from the random ones, we apply the hypothesis testing, which has been long studied in statistics.



## 2.2 Dependency Test

Under our probabilistic model about transaction data, the distribution of  $c(E)$  is Binomial with parameter  $n$  and  $p_E$ , where  $p_E$  is the probability of  $E$  being in a transaction. If the  $m$  items in  $E$  are independent,

$$p_E = P(\{i_1, i_2, \dots, i_m\} \subset T) = \prod_{j=1}^m P(i_j \subset T) = \prod_{j=1}^m p_j.$$

On the other hand, if the  $m$  items are associated, the probability of their occurring should be higher than that under the independent assumption, i.e.,  $p_E > p^*$ , where  $p^*$  denotes  $\prod_{j=1}^m p_j$ . In the later computation, all the  $p_j$ 's, whose real values are not available, will be replaced by their estimators  $\hat{p}_j = c(i_j)/n$ .

These can be further formalized as the following two hypothesis:

$$\begin{aligned} H_0 \text{ (null hypothesis)} &: p_E = p^* \\ H_a \text{ (alternative hypothesis)} &: p_E > p^*, \end{aligned} \quad (1)$$

As shown in [4], we should reject the null hypothesis (independence assumption), if  $c(E)$  is bigger than some threshold. Such a threshold can be determined by the pre-specified significance level  $\alpha$  ( $0 < \alpha < 0.5$ ), where  $\alpha$  is known as the upper bound for the probability of false positive (incorrectly accepting  $H_a$ ). The dependency test<sup>1</sup> is then as follows: given a significance level  $\alpha$ , item set  $E$  is tested to be dependent, if

$$c(E) \geq c_\alpha = \max \left\{ c : \sum_{i=c}^n \binom{n}{i} (p^*)^i (1-p^*)^{n-i} < \alpha \right\}.$$

We will denote  $c_\alpha$  by  $\text{minsup}(E)$  since it is the *minimal support* for item set  $E$  to pass the dependency test at significance level  $\alpha$ .

We note that when the sample size  $n$  is large, the above exact calculation for  $\text{minsup}(E)$  could be very time-consuming. In this case, the Normal approximation can be applied. Under the null hypothesis, the normalized  $c(E)$ ,

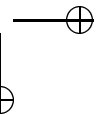
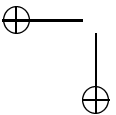
$$Z = \frac{c(E) - np^*}{\sqrt{np^*(1-p^*)}} \quad (2)$$

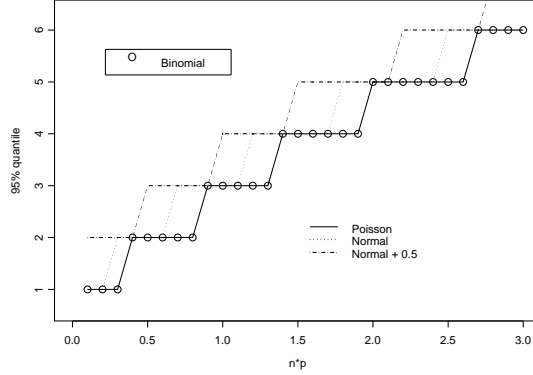
approaches to the standard Normal distribution by the Central Limiting Theorem[6]. Thus,

$$\text{minsup}(E) = np^* + z_\alpha \sqrt{np^*(1-p^*)}, \quad (3)$$

where  $z_\alpha$  is the corresponding  $1 - \alpha$  normal quantile which can be easily found in any normal table or calculated. For example,  $z_\alpha = 1.64$  for  $\alpha = 5\%$ . Intuitively,  $\alpha$  represents a confident level of the test. A commonly used value is between 1%

<sup>1</sup>The dependency test is optimal in the sense that it has the smallest probability for false negative (incorrectly accept  $H_0$ ) among all the tests with the same significance level, i.e., it is an *uniformly most powerful test*.





**Figure 1.** Comparison of different approximation for Binomial 95% quantile

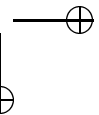
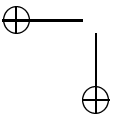
to 10%. There are many literatures discussing various modifications to improve the accuracy of the normal approximation, such as [3], [16],[17] and [15]. In our application, it is safe to assume that the number of transactions  $n$  is large. However, we still need to distinguish two cases:  $np^*$  is large and  $np^*$  is small, where the  $np^*$  represents the average number of instances of itemsets in data. When  $np^*$  is reasonable large, say  $np^* > 5$  [7][9], the usual normal approximation (2) has been shown to be sufficiently accurate. However, when  $p^*$  is extremely small, the Poisson distribution has been shown to have a better accuracy[6]. In this case,  $minsup(E)$  becomes

$$minsup(E) = \max \left\{ c : 1 - \sum_{i=0}^{c-1} \frac{e^{-np^*} (np^*)^i}{i!} < \alpha \right\}. \quad (4)$$

To further evaluate the accuracy of the two approximations, we draw in Figure 1 the 95% quantile from Binomial (denoted by circle), Poisson and Normal distributions versus  $np^*$  when  $np^*$  is less than 5, where the quantiles from normal are rounded to integers. We can see that the Poisson approximation works better than Normal. In this figure, we also plot the quantile from the normal with the continuity correction[6], i.e.  $Z = (c(E) - np^* - 0.5) / \sqrt{np^*(1 - p^*)}$ . We find that such modification actually works worse than the usual normal approximation. To switch from the Normal approximation to the Poisson, the rule of thumb used by many chi-square applications is that we switch from the Normal approximation to the Poisson when  $np^* < 5$

### 2.3 Definition and Properties of d-pattern

Ideally, we would like to find all itemsets that are dependent. However, this is computational infeasible as we show in the following that the dependency test is neither downward nor upward close.



**Property** Dependency test is neither upward nor downward closed .

**Proof:** It is enough to give two counterexamples for the normal approximation. We will consider an item set  $E$  including three items  $i_1, i_2$  and  $i_3$ .

- Counterexample for upward closeness: It's possible that  $(i_1, i_2)$  passes the dependency test, but  $c(E) = 0$ .
- Counterexample for downward closeness: Suppose that we have  $n = 20$  transactions and the observed counts are:  $c(i_1) = c(i_2) = c(i_3) = 10$  and  $c(i_1i_3) = c(i_1i_2i_3) = 4$ . Consider the item set  $E = \{i_1i_3\}$  and its superset  $F = \{i_1i_2i_3\}$ . When we set  $\alpha = 20\%$ ,  $F$  will pass the test. But  $E$  can't pass the test with any significant level  $\alpha < 50\%$ , because there dose not exist any  $z_\alpha > 0$  (corresponding to  $\alpha < 50\%$ ) satisfying  $c(E) = 4 > n\hat{p}_1\hat{p}_3 + z_s\sqrt{n\hat{p}_1\hat{p}_3(1 - \hat{p}_1\hat{p}_3)} > 5$ , observing that  $n\hat{p}_1\hat{p}_3 = 20(10/20)(10/20) = 5$ .

□

As it is computationally infeasible to discovery all itemsets that are dependent, we define a stronger dependency condition. This enables us to discover all such itemsets efficiently.

**Definition** For a given significant level  $\alpha$ , an item set  $E = \{i_1i_2 \cdots i_m\}$  ( $m \geq 2$ ) is a qualified d-pattern, if

1.  $E$  passes the dependency test with significant level  $\alpha$ ;
2. Any subset of  $E$  with more than one item also passes the dependency test with significant level  $\alpha$ .

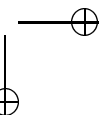
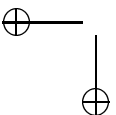
The first condition in the definition tests whether the item set is dependent or not. This differentiates a qualified pattern from a random one. The the second condition provides two benefits. First, it enables us to develop an efficient algorithm for discovering all d-patterns. Second, it avoids some false patterns. We recall the second counterexample in the proof for proposition ?? shows that it is possible that set  $\{i_1i_2i_3\}$  passes the dependency test, even though  $i_3$  is independent of the other two. The second condition of d-patterns avoids this. Further, Figure 2 shows three different Bayes Network structures for items  $a, b$  and  $c$ . No matter which model our data is generated from, the longest pattern passing the dependency test will be  $\{abc\}$ . The d-patterns are different for the three different structures:  $\{ab, ac\}$  for model (1),  $\{abc\}$  for model (2) and  $\{ab\}$  for model (3).

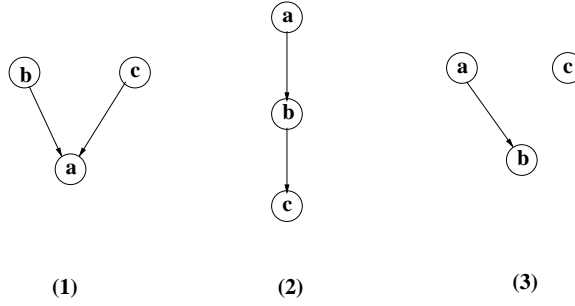
Compared with mining for frequent itemsets, in which an user defines a constant  $minsup$ , the d-patterns are qualified by a variable  $minsup(E)$ , which depends on the distributions of the items in  $E$  as well as the length of  $E$ . This naturally provides some nice properties.

**Property 1** The d-pattern is downward closed.

**Proof:** The conclusion is followed by condition (2) in the definition.

**Property 2** The minimal support  $minsup(E)$  increases as the frequency of items increases, when their product  $p^* \leq 50\%$ .





**Figure 2.** Different Bayes Network Structure for three items  $a$ ,  $b$  and  $c$ .

**Proof:** When  $p^* \leq 50\%$ , it is easy to see that  $\text{minsup}(E) = np^* + z_\alpha \sqrt{np^*(1-p^*)}$  (by normal approximation) is an increasing function of  $p^*$ . The  $\text{minsup}(E)$  for Poisson approximation is equal to the  $(1-\alpha)$  Poisson( $np^*$ ) quantile. Then, to show that  $\text{minsup}(E)$  is an increasing function of  $p^*$  is equivalent to show that for any fixed  $k$ , the cumulative probability of Poisson( $\lambda$ )

$$F(k, \lambda) = \mathbb{P}(\text{Poisson}(\lambda) \leq k) = \sum_{i=0}^k \frac{e^{-\lambda} \lambda^i}{i!}$$

is a decreasing function of  $\lambda$ . Take the derivative of  $F(k, \lambda)$  with respect to  $\lambda$ .

$$\begin{aligned} \frac{\partial}{\partial \lambda} F(k, \lambda) &= \sum_{i=0}^k \left[ -\frac{e^{-\lambda} \lambda^i}{i!} + \frac{e^{-\lambda} i \lambda^{i-1}}{i!} \right] \\ &= -\sum_{i=0}^k \frac{e^{-\lambda} \lambda^i}{i!} + \sum_{i=0}^{k-1} \frac{e^{-\lambda} \lambda^i}{i!} = -e^{-\lambda} \frac{\lambda^k}{k!} \leq 0 \end{aligned}$$

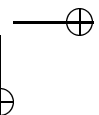
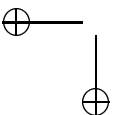
So we've shown that  $F(k, \lambda)$  is a decreasing function of  $\lambda$  with fixed  $k$ .  $\square$

In the appendix, we further show that  $p^*$ 's upper bound, 50% can be actually relaxed. For example, when the total transactions is over 100, we can set the upper bound to be 99%. However, the 50% upper bound is already enough for most of our applications.

**Property 3** The minimal support  $\text{minsup}(E)$  decreases as the size of  $E$  increases. Especially, the decrease is exponentially fast for  $\text{minsup}(E)$  by the normal approximation.

**Proof:** Recall that  $p^* = p_1 p_2 \dots p_m$ . When  $m$  increases,  $p^*$  decreases by multiplied with a number less than 1. So the result follows by the property 2. For normal approximation, the leading term  $np^*$  decays exponentially fast when  $m$  increases.

Clearly, Property 1 enables to develop an efficient algorithm. Property 2 provides a natural solution to deal with unevenly distributed data. Property 3 encourages the algorithm to discover long patterns.



## 2.4 Level-wise Algorithms

Because d-patterns are downward closed (Property 1), an level-wise search algorithm can be developed as detailed in Algorithm 1.

---

### Algorithm 2.1 Level-wise algorithm for mining d-patterns

---

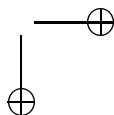
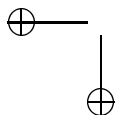
**Input:** Transaction data  $\mathcal{T}$  and significant level  $\alpha$

**Output:** all d-patterns

- (1)  $L_1 = \{\{a\} | a \in I\}$ ; Scan  $\mathcal{T}$  to count the occurrences of each pattern in  $L_1$
  - (2) Find level 2 d-patterns:  $L_2$
  - while**  $L_k \neq \emptyset$  **do**
    - (4) Generate  $C_{k+1}$  based on  $L_k$ ;
    - if**  $C_{k+1}$  is  $\emptyset$  **then**
      - break;
    - end if**
    - $k=k+1$
    - (5) Scan data to count occurrences of each candidate in  $C_k$ ;
    - (6) Find all qualified d-patterns  $L_k = \{v | v \in C_k \text{ and } \text{isDPattern}(v) \text{ is true}\}$ ;
  - end while**
  - (7) Output  $\cup_{i=2}^k L_i$
- 

Our algorithm searches the candidate space in a level-wise. This is similar to Apriori[1]. Step 1 and 2 handle the special situations for level 1 and 2. Step 4 constructs candidates  $C_{k+1}$  based on the qualified patterns  $L_k$  in the previous level. This can be accomplished by the joint operation followed by the pruning done in the A-priori algorithm[1]. Step 5 scans data and computes count for each candidate itemset. Step 6 finds all d-patterns of length  $k$  by using **isDPattern**. These steps are iterated until there is no more qualified d-patterns.

Now, we discuss some differences between our algorithm and Apriori. First, by definition, the length of a d-pattern is at least two. However, we still need to scan data and compute the count of each item. These counts will be used in the later steps. Second, we use a different criterion to qualify patterns as detailed in **isDPattern**. The basic idea is to check the count of an itemset, if the count is above 5 [9] [7], we use the normal approximation (Equation (3)); otherwise, Equation (4) is used to give more accurate estimation for an infrequent items. Last, we note that the candidate space at the second level,  $C_2$ , can not be pruned through the normal joint operation. However, we note that the level 2 itemsets can be pruned by using the upper bound. Consider any 2-itemset  $\{i_j, i_l\} \in C_2$ . As  $\text{count}(i_j, i_l) \leq \min(\text{count}(i_j), \text{count}(i_l))$ ,  $\{i_j, i_l\}$  can not pass the dependency test, if either  $\text{count}(i_j)$  or  $\text{count}(i_l)$  is not bigger than  $\text{minsup}(i_j, i_l) = n\hat{p}_j\hat{p}_l + z_\alpha\sqrt{n\hat{p}_j\hat{p}_l(1-\hat{p}_j\hat{p}_l)}$ . This upper bound allows us to conduct pruning at the second level.



---

**Algorithm 2.2** isDPattern. Algorithm for qualifying a d-pattern

---

**Input:** an item set  $E$ , and the counts of every its subsets.**Output:** True, if  $E$  is a d-pattern; False, otherwise

```
if  $count(E) \geq 5$  then
  Compute  $mincount(E)$  by Equation (3).
else
  Compute  $mincount(E)$  by Equation (4).
end if
if  $count(E) > mincount(E)$  then
  Return true
else
  Return false
end if
```

---

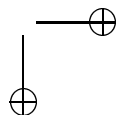
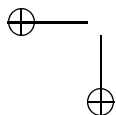
It is easy to see that Algorithm 1 needs  $k-1$  data scans. The complexity of this algorithm is linearly dependent on the length of data, but is exponentially dependent on the size of the longest d-pattern. In practice, the algorithm converges quickly, especially when patterns are not very long.

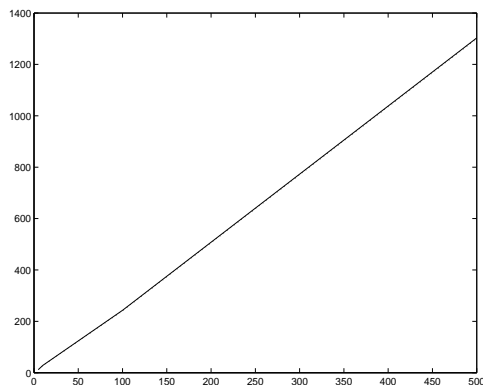
### 3 Experimental Results

This section assesses our algorithm for discovering d-patterns. Two kinds of assessments are presented. The first one evaluates the performance of our algorithm using synthetic transaction data. Here, we will demonstrate that our algorithm for d-patterns has significant advantages over that for mining frequent itemsets at the presence of noise and in the situation that items are unevenly distributed. The second one studies our algorithm using real data collected from a production computer network. Our results demonstrate that there are indeed many infrequent d-patterns that can not be discovered by the traditional approach.

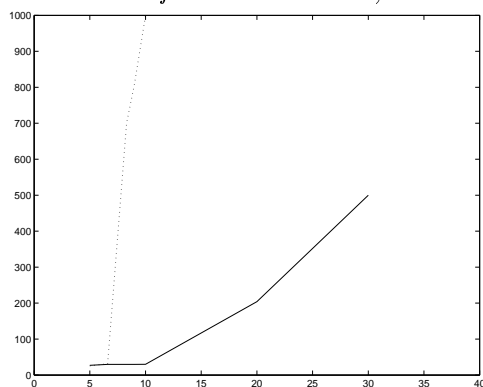
#### 3.1 Synthetic data

We begin by using synthetic data to study the scalability and efficiency of our algorithm for discovering d-patterns. The synthetic data are constructed by first generating items randomly and uniformly, and then adding instances of patterns into randomly selected transactions. Thus, the synthetic data are specified by the following parameters: the number of transactions, the number of distinct items, the average number of random items per transaction, the number of the instances of patterns and their length, and the noise to single ratio (NSR). Here, the NSR for an item in a pattern is defined by the ratio between the number of random instances to the number of the item instances in the pattern. Throughout, the





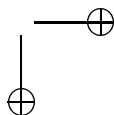
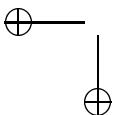
**Figure 3.** Average run time in second vs. the number of transactions in 1,000.



**Figure 4.** Average run time in second vs. NSR (noise signal ratio).

number of distinct items is 1000, the number of patterns is 5 with length 5, the average number of random items in a transaction is 10 and the NSR is 5.

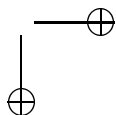
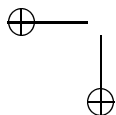
We assess scalability by varying the number of transactions and compare the level-wise algorithm for mining frequent patterns and our d-patterns. The *minchi* (i.e.  $z_\alpha$  defined in Section 2) is set to achieve 95% confidence level. The *minsup* for frequent patterns is adjusted so that there are no false positives above level 2. An experiment consists of 5 runs done with different random seeds. Figure 3 plots the average CPU time against the total number of transactions (in thousands). The results for frequent itemsets are indicated by the dotted line and those for d-patterns by the slide line. We see that the two curves are almost indistinguishable, although the curve for frequent patterns is just below that for d-patterns. This shows that the extra overhead required for computing dependency test is very small. Indeed, we see that both algorithms scale linearly as the number of transactions increases.

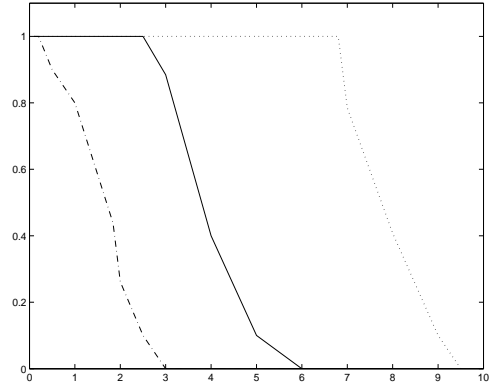


Now, we study the effect of noise by varying the NSR, and evaluate our results in terms of two types of errors: false positive and false negative. We note that we can always lower the *minchi* or *minsup* so as to discover all true patterns in the expense of generating more false positive itemsets. In our first experience, both the *minchi* for d-patterns and *minsup* for frequent itemsets are tuned so that there is no false negative. Here, the number of transaction is fixed at 10,000. The results are plotted in Figure 4. The x-axis is *NSR*, and the y-axis is the cpu run time in second. The solid line represents the results for the d-pattern algorithm, and the dotted line is for the frequent itemset algorithm. The figure shows that the cpu time of both algorithms grow exponential after some cut-off points. This is because as the NSR increases, it becomes more difficult to distinguish true patterns from noisy patterns. Thus, for a large NSR, the *minchi* threshold has to be set very low in order to discover all true patterns. However, doing so picks up more noisy patterns and thus use more cpu-time for counting. A second observation from the figure is that the d-pattern algorithm can tolerance much more noise, and its performance (so as its total candidate itemsets) decreases much more gracefully than that of the frequent itemset algorithm. This can be explained by the fact that the *minsup* of frequent itemsets should be set below the number of the instances of the true patterns. However, the *minchi* of d-patterns can be considerably larger than that of the true patterns for mining at a low level by taking into a consideration of the expected noisy instances.

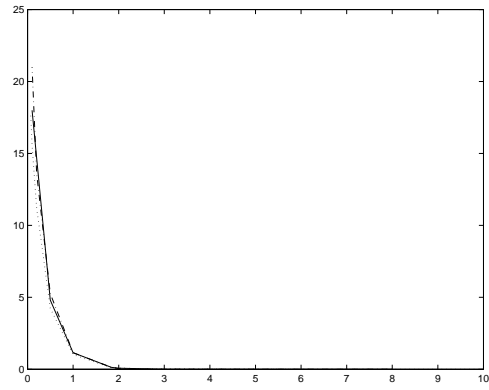
Our second experience studies how the chi-threshold relates to the false positive and the false negative of our results. Figure 5 draws false positive for NSR 20 (dashed line), 10 (solid line) and 5 (dotted line). Figure 6 shows the false negative. In these two figures, the y-axis is normalized by the total number of the subsets of true itemsets. Clearly, the smaller the *minchi*, the more false positive and the less false negative. Also, when NSR is not too large (in this case NSR is less than 15), both the false positive and negative can be close to zero. For example, when NSR is 5, we can achieve very good overall results for  $1.5 < \textit{minchi} < 7$ . However, when NSR is large, this is no longer true. For example, when NSR is 20, we either miss true patterns when *minchi* is large or result in too many false patterns when *minchi* is small. However, we note that even in this very noisy case, the d-patterns still significantly outperform the frequent patterns as we shown in Figure 4.

Third, we study the effect of unevenly distributed itemsets. Here, we divide items into two groups with 50 items each.  $10k$  transactions are randomly generated with items in the first group. We then generate  $10k \times u$  transactions for the second group of items with the NSR being 5.  $u$  represents the uneven factor. Figure 7 plots the cpu run time versus the uneven factor. It shows that the frequent itemset algorithm runs exponentially with respect to  $u$  because there are many random itemsets being above a threshold. While, the uneven factor has a little impact of the performance of the d-pattern algorithm that discover all true itemsets with *minchi* being 1.84 that corresponds to 5% statistical confidence. This results is consistent with our analytical results derived in Section 3.

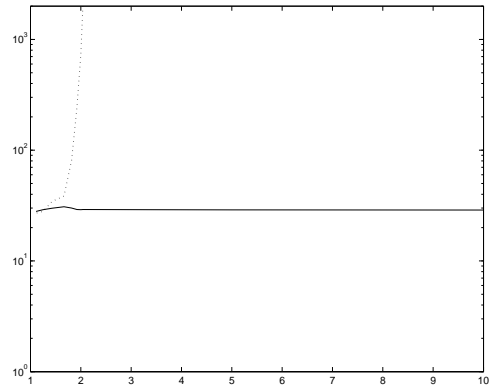




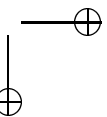
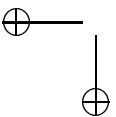
**Figure 5.** *False positive vs. minchi*



**Figure 6.** *Normalized false negative vs. minchi*



**Figure 7.** *CPU time vs. uneven factor*



## 3.2 Production Data

The windowing schemes as in [14] can construct transactions from time series data. Doing so allows us to discover temporal d-patterns<sup>2</sup>. In this section we apply the algorithms for discovering d-patterns in temporal data from a production computer network. Here, our evaluation criteria are more subjective than the last section in that we must rely on the operations staff to detect whether we have false positives or false negatives.

Two temporal data sets are considered. The first was collected from an insurance company that has events from over two thousand network elements (e.g., routers, hubs, and servers). The second was obtained from an outsourcing center that supports multiple application servers across a large geographical region. Events in the second data set are mostly server-oriented (e.g. the cpu utilization of a server is above a threshold), and those in the first relate largely to network events (e.g. “link down” events). Each data set consists of a series of records describing events received by a network console. An event has three attributes of interest here: host name, which is the source of the event; alarm type, which specifies what happened (e.g., a connection was lost, port up); and the time when the event message was received at the network console. We preprocess these data to convert events into items, where an item is a distinct pair of host and alarm type. The first data set contains approximately 70,000 events for which there are over 2,000 distinct items during a two-week period. The second data set contains over 100,000 events for which there are over 3,000 distinct items across three weeks.

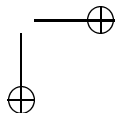
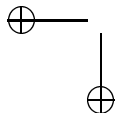
We apply our algorithm for mining d-patterns to both data sets, and compare the results to those for mining frequent itemsets. We fix *minsup* to be 3 so as to eliminate a pattern with only one or two instances. Our results are reported in Tables 3 and 4 for data sets 1 and 2, respectively.

Table 1 displays the d-patterns discovered by level (pattern size) in the level-wise search. Column 1 is the level index, i.e. the size of d-patterns; column 2 indicates the number of broader d-patterns found from the first data set. A broader d-pattern is the one that is not a subset of any other d-pattern. Column 3 is the actual count (minimum to maximum) of d-patterns at the corresponding level. Columns 4 and 5 report results for the second data set.

From the table, we can observe that there are indeed many d-patterns in our data. Over half of them have a small number of occurrences, less than 10 in both data. To provide some insights of how frequent itemsets performance, Figure 8 plots the total number of frequent itemsets (the solid line) and the number of border frequent itemsets (the dashed line) against *minsup*. Here, a border pattern refers to a pattern that is not a subset of any other pattern. The x-axis is *minsup*, and the y-axis is the number of frequent patterns discovered on a log scale. Note that the number of frequent patterns is huge—in excess of 1,000—even when *minsup* is 20. Examining the frequent patterns closely, we find that most of them can not pass the dependency test. This is not surprise since the marginal distribution of items in our data is highly skewed. Indeed, a small set of items account for over 90% of total

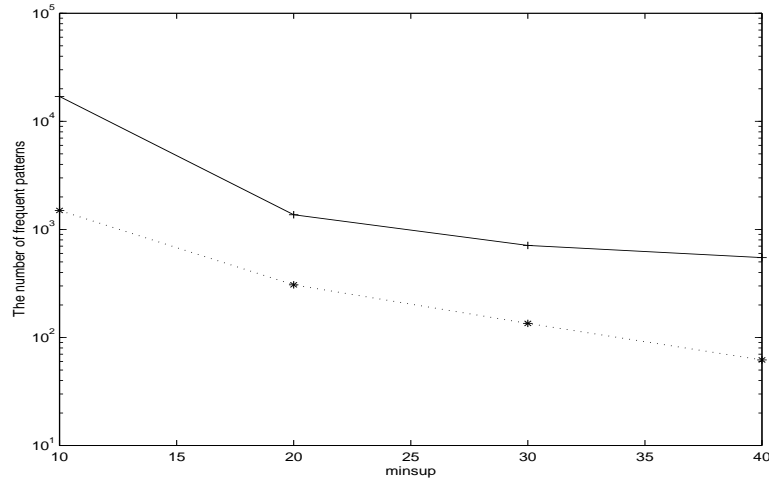
---

<sup>2</sup>Concerns are needed to deal with the overlapped windows. We refer to [14] for more detailed discussion



Level	d-patterns (Data 1)	count (min:max) (Data 1)	d-patterns (Data 2)	count (min:max) (Data 2)
2	273	3:144	160	4:137
3	56	3:8	53	5:16
4	64	3:7	77	4:61
5	25	3:7	44	5:6
6	7	3:9	41	5:9
7	0		13	4:61
8	0		11	5:6
9	0		9	5:6
10	0		2	4:5
11	0			
12	0			
13	1	3		

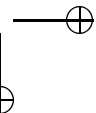
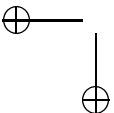
**Table 1.** *Experimental results*



**Figure 8.** *Frequent patterns of the first data set. “-”: the number of frequent patterns in the log scale; “.”: the number of border frequent patterns in the log scale; x-axis is minp*

events, and consequently, these items tend to appear in many frequent patterns. Beyond the quality of the results produced by mining for frequent itemsets, there is an issue with scalability as well. When we attempt to run with  $minsup = 3$ , more than 30k candidates are generated at the third level. Not only does this result in very large computation time, we ultimately run out of memory and so are unable to process the data.

We reviewed the d-patterns found with the operations staff. Many patterns



are related to installation errors (e.g. a wrong parameter setting of a monitoring agent) and redundant events (e.g. 11 events are generated to signal a single problem). In addition, a couple of correlations were discovered that are being studied for incorporating into event correlation rules for the real-time monitoring. We emphasize that over half of the d-patterns discovered have very low support levels that can be discovered by the traditional approach.

## 4 Conclusion

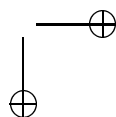
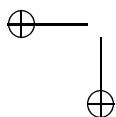
In this paper, we address three fundamental issues of the traditional association mining so as to mine patterns that are infrequent from noise, and unevenly distributed data. Our key idea is to use the dependency test. The dependency test has been used before under the large sample assumption. This work argues that it is possible to apply the dependency test for infrequent itemsets.

As the dependency test is neither downward nor upward closed, we define a strong dependency test that requires not only an itemset but also all its subsets satisfy the dependency test. Although the strong dependency test may miss some dependent patterns, a level-wise algorithm can be constructed so as to discover all d-patterns regardless of their support. To provide the ability to discover more dependent patterns that may be of interest, we further extend the d-pattern in our discussion in two ways: one is a more complicated dependency defined by both positive and negative dependency, another is combining with the frequency test so that we can discover patterns that either frequent or infrequent but fully dependent.

We demonstrate that the d-patterns significantly outperform the frequent pattern both analytically and empirically. In particular, we develop analytical results that characterize the accuracy of the d-pattern test at the noisy and lossy environment. We conduct intensive experiments using both synthetic and real-word data. Our results confirm that the d-patterns are insensitive to unevenly distributed data and work gracefully in the presence of noise.

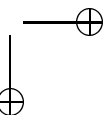
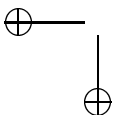
## Acknowledgments

We are grateful for Jiong Yang, Wei Wang, Irina Rish and Charles Perng's valuable discussions. Discussions with Professors John Hartigan and Andrew Barron are also really appreciated.



# Bibliography

- [1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. of VLDB*, pages 207–216, 1993.
- [2] S. Brin, R. Motiwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. *Data Mining and Knowledge Discovery*, pages 39–68, 1998.
- [3] Andrew Carter and Davi Pollard. Tusnady’s inequality revisited. Technical report, Statistics Department, Yale University, New Haven, CT, July 2000.
- [4] G. Casella and Berger R. L. *Statistical Inference*. 1990.
- [5] Edith Cohen, Mayur Datar, Shinji Fujiwara, Aristides Gionis, Piotr Indyk, Rajeev Motwani, Jeffrey D. Ullman, and Cheng Yang. Finding interesting associations without support pruning. In *ICDE*, pages 489–499, 2000.
- [6] R. Durrett. *Probability: Theory and Examples*. Duxbury Press, 1996.
- [7] Priscilla E. Greenwood and Mikhail S. Nikulin. *A Guide to Chi-squared Testing*. John Wiley & Sons, Inc., New York, NY, 1996.
- [8] J.L. Hellerstein and S. Ma. Mining event data for actionable patterns. In *International Conference for the resource management & performance evaluation of enterprive computing systems*, 2000.
- [9] H. O. Lancaster. *The Chi-squared distribution*. John Wiley & Sons, New York, 1969.
- [10] B. Liu, W. Hsu, and Y. Ma. Mining association rules wiht multiple minimum supports. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1999.
- [11] S. Ma and J.L. Hellerstein. Mining partially periodic event patterns. In *ICDE*, pages 205–214, 2001.
- [12] H. Mannila and D. Rusakov. Decomposition of event sequence into independent components. In *SAIMKDD*, 2001.



- [13] H. Mannila and J. K. Seppanen. Finding similar situations in sequences of events via random projections. In *SAIMKDD*, 2001.
- [14] H. Mannila, H. Toivonen, and A. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3), 1997.
- [15] W. Molenaar. Approximations to the poisson, binomial, and hypergeometric distribution functions. Technical Report CWI Tract 31, Center for Mathematics and Computer Science, 1970.
- [16] David B. Peizer and John W. Pratt. A normal approximation for binomial, f, beta, and other common, related tail probability, i. *Journal of the American Statistical Association*, 63:1416–1456, December 1968.
- [17] John W. Pratt. A normal approximation for binomial, f, beta, and other common, related tail probability, ii. *Journal of the American Statistical Association*, 63:1457–1483, December 1968.
- [18] J. Yang, W. Wang, and P. S. Yu. Informiner: Mining surprising periodic patterns. In *Int'l Conf. on Knowledge Discovery and Data Mining*, 2001.

