Understanding Data Completeness in Network Monitoring Systems

Flip Korn [1], Ruilin Liu, Hui (Wendy) Wang [2]

[1]*AT&T Labs–Research, Florham Park, NJ*
[2]*Department of Computer Science, Stevens Institute of Technology, Hoboken, NJ*
[1]flip@research.att.com, [2]rliu3, Hui.Wang@stevens.edu

***Abstract*—In many networks including Internet Service Providers, transportation monitoring systems and the electric grid, measurements from a set of objects are continuously taken over time and used for important decisions such as provisioning and pricing. It is therefore vital to understand the completeness and reliability of such data. Given the large volume of data generated by these systems, rather than enumerating the times and objects incurring missing or spurious data, it is more effective to provide patterns (groups of tuples) concisely summarizing trends that may not otherwise be readily observable.**

In this paper, we define the Graph Tableau Discovery Problem where the measured tuples can be thought of as edges in a bipartite graph on an ordered attribute (time) and an unordered attribute (object identifiers). We define the problem of finding an optimal summary, show that it is NP-complete, and then provide a polynomial-time approximation algorithm with guarantees to find a good summary. Experiments on real and synthetic data demonstrate the effectiveness and efficiency of our approach.

I. Introduction

In many network monitoring systems, from sensor networks to SNMP-based IP traffic measurements, a variety of objects are polled at a sequence of time points to obtain a picture of the network over time. Since the statistics obtained are used for making important decisions, from provisioning to congestion control to pricing, it is crucial that this data be reliable. However, these monitoring systems can be surprisingly sophisticated and many kinds of subtle hardware, software and logistical errors can lead to poor data quality. For example, SNMP measurements are polled using the unreliable UDP protocol, so measurements can be delayed or dropped when there is high network congestion. In sensor networks, measurements are sometimes lost due to discharged batteries or harsh environments, and delayed due to complex routing topologies.

Identifying exactly when and where significant loss (or corruption) occurs is imperative for understanding its pervasiveness as well as for diagnosing causality. When the data is collected over a large network and over a large block of time, this can be challenging. Rather than enumerating the missing tuples individually, it is more informative to present a concise summary highlighting "regions" (subsets) where the loss occurred. Here we assume correlations in time are likely (for example, due to a temporary malfunctioning of objects, pollers or routers) but make no assumptions about relationships between objects, since they can be quite complex in practice.[1]

TABLE I
EXAMPLE OF POLLED DATA

name	time	num_polls
Chicago3	10:00	1
Chicago4	10:00	1
NewYork1	10:05	1
Chicago3	10:05	1
Chicago4	10:05	1
NewYork1	10:10	1
NewYork2	10:10	1
Chicago3	10:10	1
Chicago4	10:10	1
NewYork1	10:15	1

One notion of completeness in a database, for a given set of attributes, is that the cross-product of values from their domains exists, which is classically expressed using multivalued dependencies. When data arrives in temporal order, it may be useful to consider the completeness of attribute domains as a function of time. For example, one may wish to assert that every object in the network be polled every 5 minutes. In some monitoring environments, such as IP networks during periods of congestion or sensor networks in harsh environments, occasional missing data is unavoidable; however, widespread or prolonged loss is unacceptable. Our goal is to discover and concisely summarize patterns of missing data, since clustering data that behave similarly can yield insights. Consider the relation of polled measurements given in Table I, which can be thought of as a sparse representation of the records having `num_polls=1` (the remaining tuples in the cross-product being 0's). The following three *patterns*, which are ordered pairs consisting of a time interval and object set, represent data of high loss (where at least 75% of desired measurements are missing):

Pattern 1: ([10:15, 10:15], {NewYork1, NewYork2, Chicago3, Chicago4})
Pattern 2: ([10:00, 10:00], {NewYork1, NewYork2})
Pattern 3: ([10:00, 10:15], {NewYork2})

Pattern 1 reveals a particular time (10:15) when widespread loss across all the stations in the network occurred; Pattern 2

[1]Exploiting a set of relationships between objects given *a priori* is the subject of ongoing work.

reveals a region (New York) which experienced problems at a specific time (10:00); and Pattern 3 gives a specific station (NewYork2) for which measurements are missing for the entire duration. We call a set of these patterns a *graph tableau* and, since the patterns above summarize missing tuples rather than tuples present (in this case with 75% or more loss), we call it a *fail* tableau with less than 25% *confidence*. For example, Pattern 1 covers four tuples in its cross-product (one timestamp with four stations), but only one tuple is present in Table I. Thus its confidence is measured as 1/4 = 25%. Furthermore, the above tableau summarizes that out of the sixteen tuples in the cross-product of the table (four timestamps with four stations), ten tuples are present and six tuples are missing. We say it has the *support* of 6/6, or 100%. Note that these patterns are overlapping; for example, the tuple (10:00, NewYork2) is contained in both Pattern 2 and 3. Allowing overlap is useful in practice since loss may be due to multiple (overlapping) causes, thus enabling the summary to find the right fit.

In general, graph tableaux are capable of illuminating the distribution of loss, such as whether it is concentrated in time (perhaps due to a temporary problem such as a poller outage) or over a set of stations (perhaps due to a misconfigured network topology). In this paper, we define and provide a solution for the problem of discovering graph tableaux. In particular, we make the following contributions:

- We formalize the graph tableau discovery problem, and prove that it is NP-hard to find an optimal tableau.
- We develop a polynomial-time approximation algorithm with quality guarantees compared to an optimal tableau. We also give a tuning parameter ϵ which allows accuracy to be traded for performance.
- We perform an extensive set of experiments of our techniques on both synthetic and real data. Besides evaluation of efficiency and precision/recall of our algorithm, we study the utility of discovered tableaux, with a case study visualizing and interpreting the patterns. Our results show that the algorithms can efficiently discover meaningful graph tableaux.

The rest of the paper is organized as follows. In Section II we give problem definitions and the proof of NP-completeness. In Section III, we present a greedy algorithm to discover graph tableaux. Section IV presents an experimental evaluation of our methods. We discuss the related work in Section V, and conclude our paper in Section VI.

II. Problem Definition

Let G be a bipartite graph on vertex sets $\mathcal{T}$ and $\mathcal{O}$, with $\mathcal{T}$ denoting time stamps and $\mathcal{O}$ denoting objects. Without loss of generality, let $\mathcal{T}$ be the totally ordered set of numbers $\{1, 2, ..., n\}$ and $\mathcal{O}$ be a set of object identifiers. Let $E \subseteq \mathcal{T} \times \mathcal{O}$ be the set of tuples ("edges") obtained from the monitoring system as measurements. The ideal expectation is that every object is polled regularly at each time; that is, E should be equal to the entire cross-product $\mathcal{T} \times \mathcal{O}$. We can measure the degree to which this is satisfied, which we call the *confidence*, as the percentage of expected measurements that were obtained. We use this definition because it is simple and because it follows the measures used in related work [15]. Clearly, it is reasonable to assume that the amount of loss is proportional to the amount of data. Moreover, this definition is equivalent to the weighted average degree of the nodes, which is appropriate for Gaussian loss and also mitigates the impact of delays that may be mistaken for loss.[2] We seek "regions" of the data for which the confidence is either high (where the expectation holds) or low (where the expectation fails).

Definition 2.1: Given thresholds for confidence $\hat{c}$ and support $\hat{s}$, the *Hold Tableau Discovery Problem* is to find a smallest size set T of pairs (I, S) with "interval" $I \subseteq \mathcal{T}$ (that is, the nodes in I must be a contiguous subset of $\mathcal{T}$) and set $S \subseteq \mathcal{O}$ such that $conf(I, S) \geq \hat{c}$ for each (I, S) in T and $\left|\bigcup_{(I,S)\in T}((I \times S) \cap E)\right| \geq \hat{s}|E|$, where

$$conf(I, S) = \frac{|(I \times S) \cap E|}{|I||S|}.$$

For exposition, $|T|$ denotes the size of T, which is defined as the total number of pairs in T. However, the discussion below can be easily enhanced to handle *weighted objective functions* such as $\sum_{(I,S)\in T}(1 + |S|)$.

Definition 2.2: Let $\bar{E} = (\mathcal{T} \times \mathcal{O}) - E$. Given thresholds for confidence $\hat{c}$ and support $\hat{s}$, the *Fail Tableau Discovery Problem* is to find a smallest size set T of pairs (I, S) with (interval) $I \subseteq \mathcal{T}$ and $S \subseteq \mathcal{O}$ such that $conf(I, S) \leq \hat{c}$ for each (I, S) in T and $\left|\bigcup_{(I,S)\in T}((I \times S) \cap \bar{E})\right| \geq \hat{s}|\bar{E}|$, where

$$conf(I, S) = \frac{|(I \times S) \cap E|}{|I||S|}.$$

Proposition 2.1: The Tableau Discovery Problem is NP-complete.

The proof is in Appendix A.

Since tableau discovery is NP-hard, we will resort to a greedy approximation. Unfortunately, we cannot simply apply the well known greedy heuristic for Partial Set Cover here, since the number of candidate sets is $\Theta(|\mathcal{T}|^2 2^{|\mathcal{O}|})$. The challenge we tackle in this paper is how to efficiently find an appropriate set for each iteration of the greedy algorithm without examining all the candidates.

III. Algorithm

We assume that the complete set of object identifiers $\mathcal{O}$ and polling times $\mathcal{T}$ are known (or given first as input). In addition, the set of polled tuples $E \subseteq \mathcal{T} \times \mathcal{O}$ is given as input. We assume E is sorted lexicographically on $(\mathcal{T}, \mathcal{O})$; if not, that can be achieved efficiently since the tuples are generated by the system in non-decreasing time order. Algorithm 1 gives the algorithm pseudocode for the Hold Tableau Discovery Problem. At each iteration we wish to find a pair (I, S), where $I \subseteq \mathcal{T}$ is a contiguous set ("interval") and $S \subseteq \mathcal{O}$ is an arbitrary subset of $\mathcal{O}$, with highest "marginal support", that

[2] Adding additional minimum degree constraints on the nodes (for a "trimmed mean") requires a small modification to the algorithm.

is, having the largest number of previously uncovered edges by T; and such that the fraction of edges in this subgraph is high enough in the case of hold tableaux, or low enough in the case of fail tableaux. Let $w(t,o) \in \{0,1\}$ indicate the benefit obtained from adding edge (t,o) to T, that is,

$$w(t,o) = \begin{cases} 0, (t,o) \in T \\ 1, \text{otherwise} \end{cases}$$

Let $w(I,S)$ denote the total benefit from pair (I,S), that is, $w(I,S) = \sum_{(t,o)\in(I\times S)\cap E} w(t,o)$. Then we seek $\arg\max_{I\in\mathcal{T},S\in\mathcal{O}} w(I,S)$ having $conf(I,S) \geq \hat{c}$.

Suppose we fix some $I \subseteq \mathcal{T}$. (We will wind up trying all possible I, which is achieved by iterating through all $i = 1,..,n$ and $j = i,...,n$ in the pseudocode.) Then we can find $S_I \subseteq \mathcal{O}$ having the largest $w(I,S_I)$ as follows. Let $deg^I(o)$ denote the number of edges between $o \in \mathcal{O}$ and some element of I, i.e. $deg^I(o) = |E \cap (I \times \{o\})|$, computed in Lines 10–12 of the pseudocode. First, we select all $o \in \mathcal{O}$ having $deg^I(o) \geq \hat{c}|I|$ (shown in Lines 14–17 of the pseudocode). Notice that these nodes can only increase the confidence to $\hat{c}$ or above, so it would be imprudent not to choose them. Let S' be the subset of such nodes; so S' and $\mathcal{O} - S'$ form a partition of nodes in $\mathcal{O}$ into those which individually satisfy confidence and those which do not. Let $B = \sum_{o\in S'}(deg^I(o) - \hat{c}|I|)$. Now, for each node $v \in V_2 - S'$, compute its cost C_o as $deg^I(o) - \hat{c}|I|$. We want to choose a subset $Q \subseteq \mathcal{O} - S'$ with maximum weighted sum over its edges to I subject to the given cost budget constraint B, that is, $\sum_{o\in Q} C_o \leq B$ (shown in Lines 18–24 of the pseudocode). This is an instance of 0-1 KNAPSACK. Since the maximum total budget is bounded by $|E|$, we can solve this problem via dynamic programming in $O(|\mathcal{O}||E|)$ time. Instead, we use a modified greedy algorithm, which selects the items by value per unit cost while keeping within budget (and then considers taking the most profitable item as an alternative), since it provides a 2-approximation and can be run in $O(|\mathcal{O}|)$ time [12].[3]

Theorem 3.1: Algorithm 1 yields a $(2 + \ln|E|)$-approximation to the optimal cover size.

Proof: See [3]. ∎

At each iteration of the greedy algorithm, we can use the approach above for every possible $I \subseteq \mathcal{T}$ to find an S_I and choose a pair (I,S_I) having the largest $w(S,I)$, which is maintained in Lines 26–27 of the pseudocode; Lines 30–31 adjust the edge weights after selecting this pair. Unfortunately, there are $O(|\mathcal{T}|^2)$ different I's, making the running time $O(|\mathcal{T}|^2|\mathcal{O}|)$ per iteration.

Lemma 3.1: Algorithm 1 terminates in at most $|\mathcal{T}|(1 + \ln|E|)$ iterations.

Choosing all singleton intervals from $\mathcal{T}$ (and their edge-linked node sets from $\mathcal{O}$) is clearly a feasible solution and, therefore, the greedy algorithm must be within $1 + \ln|E|$ of this.

If we are willing to allow a $(1+\epsilon)$-approximation on confidence, where ϵ is the user-specified approximation threshold, then we only need to test $O(|\mathcal{T}|\log_{1+\epsilon}|\mathcal{T}|)$ different intervals $[i,j]$ as follows: for each $i \in \mathcal{T}$, take exponentially increasing sized intervals, $(1+\epsilon)^h$ for $h = 1,...,\lceil \log_{1+\epsilon}|\mathcal{T}| \rceil$, and find the largest integer j such that $j - i + 1 \leq (1+\epsilon)^h$.

[3]In practice, the modified greedy algorithm is quite accurate and it is only pathological cases that result in a factor of 2.

Algorithm 1 Tableau Discovery

Require: $G = (\mathcal{T}, \mathcal{O}, E)$; thresholds $\hat{c}$, $\hat{s}$; tolerance ϵ
Ensure: T with $conf(I,S) \geq \hat{c}$ for all $(I,S) \in T$ and $\sum_{(I,S)\in T} w(I,S) \geq \hat{s}|E|$

```
 1: for all (t,o) ∈ E do
 2:     w(t,o) ← 1
 3: T ← ∅, cumsupp ← 0
 4: while cumsupp < ŝ|E| do
 5:     for all i from 1 to n do
 6:         for all o ∈ O do
 7:             deg(o) ← 0
 8:         for all j from i to n do
 9:             I ← [i,j]
10:             for all o ∈ O do
11:                 if (j,o) ∈ E then
12:                     deg(o) ← deg(o) + 1
13:             S' ← ∅, Q ← ∅, B ← 0
14:             for all o ∈ O do
15:                 if deg(o) ≥ ĉ|I| then
16:                     S' ← S' ∪ {o}
17:                     B ← B + deg(o) − ĉ|I|
18:             for all o ∈ O − S' do
19:                 C_o ← ĉ|I| − deg(o)
20:             C_sum ← 0
21:             while C_sum ≤ B do
22:                 p ← arg max_{o∈O−S'} w(I,{o})/C_o
23:                 Q ← Q ∪ {p}
24:                 C_sum ← C_sum + C_p
25:             S_I ← S' ∪ Q
26:             if w(I,S_I) > w(I*,S_I*) then
27:                 (I*,S_I*) ← (I,S_I)
28:     T ← T ∪ {(I*,S*)}
29:     cumsupp ← cumsupp + w(I*,S_I*)
30:     for (t,o) ∈ (I* × S_I*) ∩ E do
31:         w(t,o) ← 0
```

Theorem 3.2: 1) (No false positives) If the algorithm outputs a pair (I,S_I), then $conf(I,S_I) \geq \hat{c}/(1+\epsilon)$.
2) (No false negatives) Let $(I',S_{I'})$ be a pair with $conf(I',S_{I'}) \geq \hat{c}$. Let I be an interval sharing the same left endpoint as I' such that $|I'| \leq |I| \leq |I'|(1+\epsilon)$. Then $conf(I,S_I) \geq \hat{c}/(1+\epsilon)$.

Proof: The first part is trivial, as it is obvious from the algorithm that if the output includes a pair (I,S_I), then $conf(I,S_I) \geq \hat{c}/(1+\epsilon)$. Modulo the distinction between $\hat{c}$ and $\hat{c}/(1+\epsilon)$, there are no "false positives".

For the second part, $conf(J,S_J) = \frac{|(J\times S_J)\cap E|}{|J||S_J|} \geq \hat{c}$. Now consider $conf(I,S_I)$. In the worst case, the additional nodes in $I - J$ provide no new edges, making the numerator as small as possible, and $|I| = |J|(1+\epsilon)$, making the denominator as large as possible. In this case, $conf(I,S_I) \leq |(J \times S_J) \cap E|/(1+\epsilon)|J||S_J| = conf(J,S_J)/(1+\epsilon) \geq \hat{c}/(1+\epsilon)$. ∎

The overall running time of this approximation at each iteration of the greedy set cover algorithm is therefore $O(\frac{1}{\epsilon}|\mathcal{T}||\mathcal{O}|\log|\mathcal{T}|)$.

Algorithm 1 can be easily modified for the Fail Tableau

Discovery problem by converting the bigraph to its negation (consisting of missing edges only) and searching for the hold tableau with confidence $conf \leq \hat{c}$ on the converted bigraph. In this case, we will use $\frac{T \setminus (I \times S)}{(|S|+1)}$ to select a pattern (I, S) at each step of the greedy algorithm to maximize the benefit per unit cost.

IV. Experiments

We did an extensive set of experiments including: (1) the evaluation of precision and recall of graph tableaux discovered by our algorithm (compared with *ground truth patterns*); (2) visualization of and interpretation based on graph tableaux applied to real data; (3) comparison with other methods in the literature; and (4) running time analysis of our algorithm under various settings. All our experiments find fail tableaux only.

A. Setup

All experiments were evaluated on a desktop PC with a 2.4GHz Intel Core 2 Quad Q6600 CPU and 3GB RAM running Ubuntu 11.10. We implemented our algorithms in C++.

We used two real data sets:

- *Caltrans road traffic data*[4], generated by embedded sensors for cars passing over the detectors scattered in California. Here $\mathcal{T}$ is the set of timestamps and $\mathcal{O}$ is the set of sensor IDs. Normally the detectors send collected data to a central station every 30 seconds. However, not all 25,000 detectors work properly at each time. Figure 1(a) gives a summary of the status of detectors on 2011/06/01, taken from the *Caltrans* website, showing that only 71.3% of them worked properly; Figure 1(b) provides possible reasons for the malfunctions.
- *NCDC Global Summary of Day (GSOD) data*[5], recording the climate measurements (temperature, sea level, etc.) collected from over 9000 worldwide stations. We chose various subsets from years 2009-2011 and from 100-5000 stations.

We also used synthetic data; the details of the generator for this data is discussed in Appendix B.

In the remaining of the section, we use the notation $dataset\_X\_Y$ for the dataset of $|\mathcal{T}|$= X and $|\mathcal{O}|$=Y.

We measure the quality of a discovered pattern using standard *precision* and *recall* metrics. First define the intersection size between two patterns $P_i = (I_i, S_i)$ and $P_j = (I_j, S_j)$ as $|I_i \cap I_j| \times |S_i \cap S_j|$. For example, if $P_1 = [2,4] \times \{1,2,3\}$ and $P_2 = [1,2] \times \{1,3\}$, then $|I_1 \cap I_2| \times |S_1 \cap S_2| = 1 \times 2 = 2$.

Given ground truth patterns $\{P_1^*, P_2^*, \cdots, P_k^*\}$ and discovered patterns $\{P_1, P_2, \cdots, P_l\}$, we define the *most relevant discovered pattern to* P_i^* as

$$MaxR(P_i^*) = \arg\max_{\forall P_j} (|I_i^* \cap I_j| \times |S_i^* \cap S_j|)$$

[4]http://pems.dot.ca.gov/

[5]ftp://ftp.ncdc.noaa.gov/pub/data/gsod/

Id	I	# missing edges	S
0	[41,74]	27	16 28 40 43
1	[87,102]	57	3 4 7 9 15 16 18 19 23 24 26 28 31 38 39 45 47 50
2	[158,169]	2	29
3	[214,219]	31	1 7 8 9 11 12 13 15 16 17 19 22 25 27 28 29 31 34 35 38 40 44 45 47 48 49
4	[241,266]	10	7 16

TABLE II
THE GROUND TRUTH PATTERNS OF SYNTHETIC DATA

and the *most relevant ground truth pattern to* P_i as

$$MaxR(P_i) = \arg\max_{\forall P_j^*} (|I_i^* \cap I_j^*| \times |S_i \cap S_j^*|)$$

For example, consider ground truth patterns $P_1^* = ([2,4], \{1,2,3\}), P_2^* = ([1,2], \{1,2,3\})$ and discovered patterns $P_1 = ([1,2], \{1,3\}), P_2 = ([2,4], \{2,3\}), P_3 = ([4,5], \{2,3,4\})$. $MaxR(P_1^*) = P_2$, $MaxR(P_1) = P_2^*$.

The *precision* of a discovered pattern $P = (I, S)$ is

$$precision(P) = \frac{|MaxR(P) \cap P|}{|I| \times |S|},$$

The *recall* of a ground truth pattern $P^* = (I^*, S^*)$ is

$$recall(P^*) = \frac{|MaxR(P^*) \cap P^*|}{|I^*| \times |S^*|}.$$

Given a set of ground truth patterns $\{P_1^*, P_2^*, \cdots, P_k^*\}$, and a set of discovered patterns $\{P_1, P_2, \cdots, P_l\}$, the *average precision* is

$$precision_{avg} = \frac{\sum_{i=1}^{l} precision(P_i)}{l},$$

and the *average recall* is

$$recall_{avg} = \frac{\sum_{i=1}^{k} recall(P_i^*)}{k}$$

B. Quality of Tableaux

We generated synthetic data as follows. The input to the generator includes $|\mathcal{T}|$, $|\mathcal{O}|$, the number of ground truth patterns k, a confidence threshold $\widehat{c}$, α for Zipf distributed random number generator, and β as the ratio of $\mathcal{T}$ nodes that will be used to create ground truth patterns. The output of the algorithm are k ground truth patterns whose intervals do not overlap. (Details of the algorithm can be found in Appendix.) We used this to generate data with $|\mathcal{T}| = 300$ and $|\mathcal{O}| = 50$ and five ground truth patterns with confidence $\widehat{c} = 0.8$, which are shown in Table II.

We set $\varepsilon = 0$ and $\widehat{s} = 1$ but vary $\widehat{c}$ from 0.55 to 0.99. The result is shown in Figure 2 (a). Observe that $\widehat{c} = 0.8$, the same $\widehat{c}$-value used to generate the data, is an inflection point in the plot. The precision increases as $\widehat{c}$ grows to 0.8 (with the recall remaining at 1) since a smaller number of missing edges can be tolerated, thus leading to more accuracy. In examing the results, we found that each ground truth pattern was contained in at least one discovered pattern (i.e., $I^*i \subseteq I_j$ and $S_i^* \subseteq S_j$ for some i and j), which explains why the recall

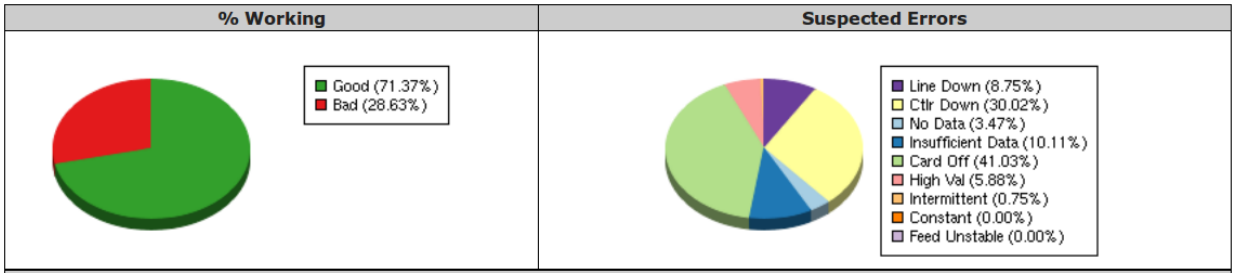
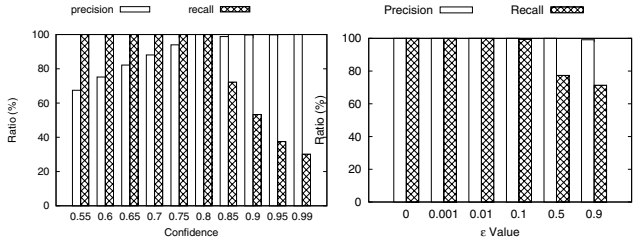


Fig. 1. Summary of Caltrans Data Quality

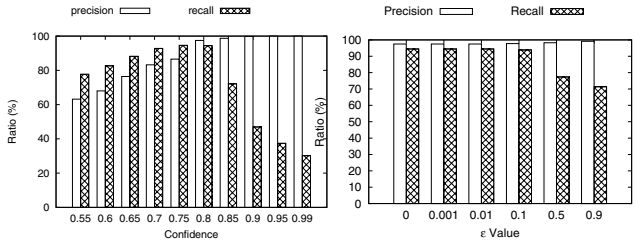


(a) $\widehat{c} = 0.55 \rightarrow 0.99$ ($\varepsilon = 0$) (b) $\varepsilon = 0.001 \rightarrow 0.9$ ($\widehat{c} = 0.95$)

Fig. 2. Precision and recall of unweighted patterns (SYN_T300_S50 dataset)

was 1 for $\widehat{c} \leq 0.8$. However, the recall starts decreasing when $\widehat{c} > 0.8$. This is because fewer missing edges are tolerated at high $\hat{c}$, so the ground truth patterns are no longer contained in a discovered pattern. For $\widehat{c} > 0.8$, the precision value is 1 because each discovered pattern was contained in at least one ground truth pattern, which is symmetric to the recall for $\widehat{c} \leq 0.8$.

We fixed $\widehat{c} = 0.8$, $\widehat{s} = 1$ and tested various ε values; Figure 2 (b) shows the results. For $\varepsilon < 0.1$, the results were exactly the same as for $\varepsilon = 0$. With increasing ε, the recall degrades, as the approximation algorithm tests fewer candidates and thus finds less tight-fitting patterns. However, the approximation does not hurt the quality of discovered tableaux by much. For instance, when $\varepsilon = 0.5$, the precision was still 1 and the recall was 0.9346. Even at $\varepsilon = 0.9$ the average precision is still higher than 0.98 and the recall is higher than 0.88. As we shall see in Section IV-E, a moderate value of ϵ brings significant improvement in running time.



(a) $\widehat{c} = 0.55 \rightarrow 0.99$ ($\varepsilon = 0$) (b) $\varepsilon = 0.001 \rightarrow 0.9$ ($\widehat{c} = 0.95$)

Fig. 3. Precision and recall of weighted tableau patterns (SYN_T300_S50 dataset, $\varepsilon = 0$)

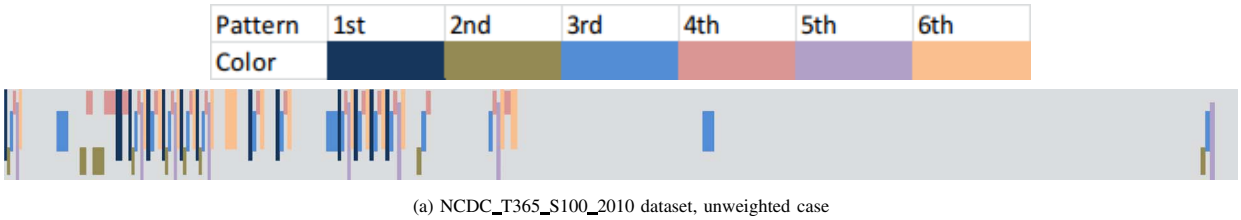
We also measure the precision and recall of weighted patterns. First, we fix $\varepsilon = 0$ and vary $\widehat{c}$ from 0.55 to 0.99. Figure 3 (a) illustrates the result. It is easy to observe that the weighted patterns follow the similar patterns as unweighted patterns (Figure 2 (a)). The only difference is that the recall of weighted patterns is always less than 1, while the recall of unweighted patterns can achieve 1, when $\widehat{c} < 0.8$. It is expected that the weighted tableaux patterns are of recall less than 1 (i.e., deviate from the ground truth patterns), since the weighted tableaux patterns are generated by using marginal *weighted* support, while the ground truth patterns are generated by using the marginal support. Despite this, the precision of weighted tableaux patterns is still high (at least 80%). We also fix $\widehat{c} = 0.95$ and vary ε from 0.001 to 0.9. The result (Figure 3 (b)) shows the similar observation as the unweighted tableaux (Figure 2 (b)), except that the recall of the weighted tableaux is always less than 1.

C. Visualization and Interpretation of Tableaux

We display the top-6 patterns using colored rectangles in Figure 4, where the x-axis denotes time and the y-axis is (lexicographically) sorted by object identifier. Each pattern (I, S) is represented by a disjoint set of thin vertical rectangles of the same color; different patterns have different colors. We used NCDC_T365_S100_2010 and NCDC_T365_S100_2011 data, which have the same stations at times during year 2010 and 2011. (We randomly chose 100 stations that appeared in both 2010 and 2011 data.)

First we considered the unweighted case (Fig. 4 (a) and (b)). In year 2010, the time interval with most loss was $[01/01/2010, 10/14/2010]$, which is covered by four of the six patterns together yielding support 0.68. In year 2011, the lossiest time interval is $[09/23/2011, 12/31/2011]$ (covered by 3/6 patterns having total support 0.42). There are other time periods, e.g., between January and March of year 2011. These periods do not contribute as much as the lossiest ones. We took the stations that appeared in the top-6 patterns and joined their IDs with the station list of NCDC dataset[6]. In all, there were 30 stations from the top-6 patterns. We found out there is exactly one station in Afghanistan which appears in every pattern. We suspect that this station may not be functioning due to volatility of that region from war. The remaining stations

[6]ftp://ftp.ncdc.noaa.gov/pub/data/inventories/ISH-HISTORY.TXT



(a) NCDC_T365_S100_2010 dataset, unweighted case

(b) NCDC_T365_S100_2011 dataset, unweighted case

(c) NCDC_T365_S100_2010 dataset, weighted case

(d) NCDC_T365_S100_2011 dataset, weighted case

Fig. 4. Tableaux Visualization ($\widehat{c} = 0.9$, $\varepsilon = 0$)

were mostly from polar regions, such as Northern Norway, with harsh climates; not surprisingly, most of the missing data was summarized by patterns containing time interval Sept-Dec as well as Jan-March, for both 2010 and 2011. In addition, we observed that no data was collected for some of these stations during the summer of 2010. We looked at these stations and determined that they were not actually set up until Sept 2010.

We also plotted the patterns for the weighted case (Fig. 4 (c) and (d)). Observe that although the discovered patterns are different from those discovered in the unweighted case (e.g, the lossiest time interval for year 2010 was from 01/01/2010-08/29/2010), there exists large overlap between these two sets of patterns. This shows that although the patterns for both weighted and unweighted cases catch different semantics, they catch the same underlying data that is of poor quality.

In addition, we analyzed the *Caltrans* traffic data, hoping to understand detector failures with respect to times and locations of measurements. We computed the top-6 patterns from a 4-hour data set (from midnight to 4am on 2011/06/01) covering 921 detectors scattered over 12 freeways around Sacramento, using different parameters ($\widehat{c} = 0.9 \rightarrow 1.0$, $\varepsilon = 0 \rightarrow 0.1$). The results include various small sets (of sizes 11-16) of detectors missing measurements over long time intervals, from 50 minutes to 4 hours, as well as one pattern containing a single timestamp (at midnight) for which 98% of the detectors had no measurement. Figure 5 shows the stations based on the patterns representing those that failed to report over long time intervals, plotted using Google Maps. For example, one of the detectors that did not report any measurements was located at NB Sunrise Boulevard, Rancho Condova. It appears that most of these detectors are located on Route 50 (from Sacramento to Placerville), Route 65 (from Yuba City to Lincoln), I-70 (from Yuba City to Sacramento) and Route 99 (from Sacramento to Elk Grove). (In contrast, the more reliable detectors were located on I-80 from Auburn to Fairfield, I-5 from Woodland to Sacramento, and Routes 28, 51, 89, 113, 160 and 275.)

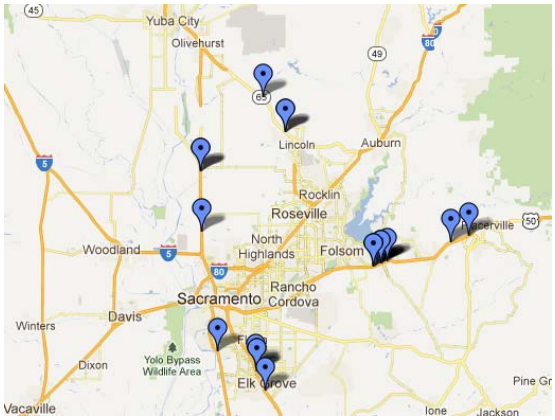
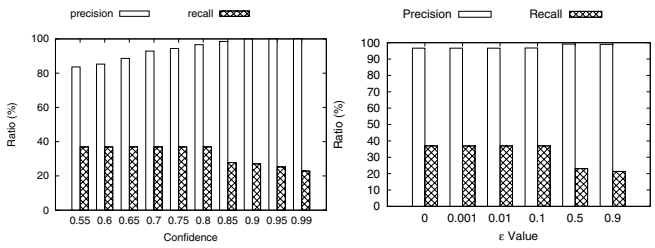


Fig. 5. Caltrans detectors missing data, based on patterns ($\widehat{c} = 0.9, \varepsilon = 0$)



(a) $\widehat{c} = 0.55 \rightarrow 0.99$ (b) $\varepsilon = 0.001 \rightarrow 0.9$

Fig. 6. Range tableaux VS. graph tableaux w.r.t quality

D. Comparison with Range Tableaux

Most of the literature on summarizing a bipartite graph using dense subgraph patterns either assumes both attributes

are ordered (e.g., [11]) or both attributes are unordered (e.g., [15]). The only existing work we are aware of which considers the case of one ordered and one unordered attribute is *range tableaux*, which was described in [8]. We compared the quality of our graph tableaux with range tableaux adapted to our scenario, which yields patterns of the form $([i,j],\mathcal{O})$ or $([i,j],\{o\})$, where o is a single object from $\mathcal{O}$.

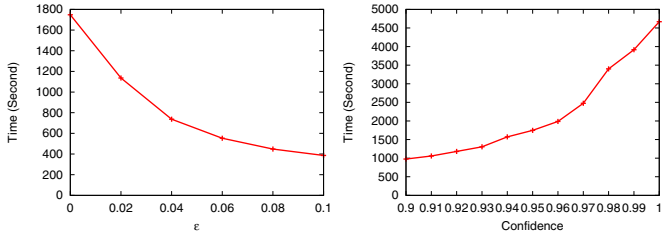
First, we compare precision and recall of the two methods using synthetic data. Figure 6 shows that the recall of the range tableaux is consistently less than 50%, and is much worse than the recall of graph tableaux (see Figure 2). This is because all the range tableaux patterns are of the form $([i,j],\{o\})$, which only covers a small portion of the ground truth patterns. On the other hand, the range tableaux patterns have high precision (no less than 0.9), since the majority of the range tableaux patterns are contained in at least one ground truth pattern. In fact, each range tableaux pattern either is contained in or intersects at least one graph tableaux pattern.

For better understanding of the difference between range tableaux and our graph tableaux, we also compare the *support ratio* and *aspect ratio* of the range tableaux with those of our graph tableaux. First, we define the *support ratio* $SR_i = \frac{|E_i|}{|E|}$, where $|E_i|$ is the marginal support of the i^{th} tableau and $|E|$ is the total number of edges of D. We compared the graph tableaux and range tableaux in NCDC_T365_S100_2010 dataset. We observed that for the unweighted case, the first 6 graph tableaux patterns contribute a significant portion to the support. For instance, when $\widehat{c} = 0.9$ and $\varepsilon = 0$, the top-6 graph tableaux patterns, which takes 6.74% of all patterns, contribute to 70% of support. Such phenomenon also holds for other $\widehat{c}$ values. However, the top-k range patterns of the largest marginal support only contribute to 15% of tableaux support. In other words, when returning the same number of patterns, our graph tabelaux is of much higher support (and thus much better quality) than the range tableaux.

Second, we examine the aspect ratio of range tableaux. We define the *aspect ratio* of the pattern $P = I \times S$ as $AR = \frac{|I|}{|S|}$. Generally speaking, small aspect ratio indicates that there are a large number of stations do not work in a short time interval, while large aspect ratio indicates that a small number of stations do not work in a large time interval. We calculate the aspect ratios of graph tableux and range tableaux patterns on NCDC_T365_S100_2010 dataset. It shows that the aspect ratio of our graph tableaux patterns span over the range $[0.02, 50]$, but the aspect ratio of range tableaux is in the range $[1, 350]$. The reason that the range tableaux is biased towards patterns of large aspect ratio is that most of the discovered range tableaux patterns consist of a single object in S.

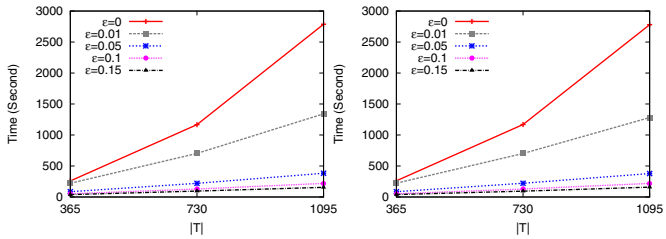
E. Performance Evaluation

In this section, we measure the running time of our tableau discovery algorithm. First, we measured the running time to achieve 100% support. We used the NCDC_T365_S100_2010 dataset; results are shown in Figure 7. For fixed $\widehat{c}$, the running time decreased with increasing ε, since larger ε allows the algorithm to test fewer intervals. On the other hand, for fixed



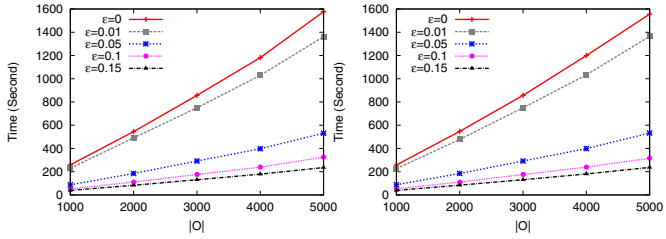
(a) $\varepsilon = 0 \rightarrow 0.1$ ($\widehat{c} = 0.95$) (b) $\widehat{c} = 0.9 \rightarrow 1$ ($\varepsilon = 0$)

Fig. 7. Time performance of discovering all tableaux (NCDC_T365_S100_2010 dataset, unweighted case)



(a) Unweighted (b) Weighted

Change $|\mathcal{T}|$, $\varepsilon = 0 \rightarrow 0.15$ ($\widehat{c} = 0.95$, $|\mathcal{O}| = 1000$)



(c) Unweighted (d) Weighted

Change $|\mathcal{O}|$, $\varepsilon = 0 \rightarrow 0.15$ ($\widehat{c} = 0.95$, $|\mathcal{T}| = 365$)

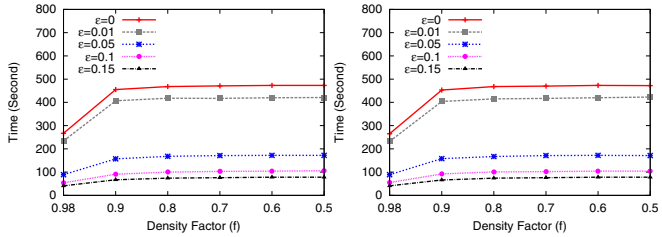
Fig. 8. Time performance of discovering top-10 tableaux on NCDC datasets

ε the running time increased as $\widehat{c}$ grew due to higher $\widehat{c}$ resulting in more tableaux patterns (each of smaller size). In these experiments, the number of patterns increased from 92 ($\widehat{c} = 0.9$) to 419 ($\widehat{c} = 1$).

To make the running time less output-sensitive, we then ran the algorithm to discover only the first 10 patterns in the remaining experiments.

Fixing the size of $\mathcal{O}$, we varied the size of $\mathcal{T}$ and measured the running time using NCDC_T365_S1000, NCDC_T730_S1000, and NCDC_T1095_S1000 datasets. Figure 8 (a) & (b) shows the result when we vary ε. For both unweighted and weighted cases, the time increased superlinearly with increasing $\mathcal{T}$ (recall that the time depends quadratically on $\mathcal{T}$ with $\epsilon = 0$ in the algorithm analysis). Apparently the weight function did not influence the running time much as it was very similar for both weighted and unweighted cases. Increasing ε by even a small amount improved the running time significantly. For example, when $|\mathcal{T}| = 1095$, increasing ε from 0 to 0.15 saved 94.28% time.

We also varied $\widehat{c}$ from 0.7 to 1.0 while fixing $\varepsilon = 0$. This



(a) Unweighted (b) Weighted

Change density factor, $\varepsilon = 0 \rightarrow 0.15$
($\widehat{c} = 0.95$, $|\mathcal{T}| = 365$, $|\mathcal{O}| = 1000$)

Fig. 9. Time performance of discovering top-10 tableaux on modified NCDC datasets

has very little impact on the running time. We omit the results due to limited space.

Fixing the size of $\mathcal{T}$, we then varied the size of $\mathcal{O}$, from 1000 to 5000 stations. We measured the running time for both the unweighted and weighted cases. Figure 8 (c) and (d) shows the results. The running time increased linearly in $|\mathcal{O}|$. and decreased with larger ε (with fixed $\widehat{c}$), which demonstrates the performance gain from relaxing the confidence bound. (We also varied $\widehat{c}$ with fixed ε but observed very little impact on running time, so omit the results.)

We also ran these experiments on the *Caltrans* data; the results yielded the same observations as the NCDC dataset so we omit them for brevity.

We studied the impact of "edge" density on tableau discovery performance by randomly removing points from the NCDC_T365_S1000 data and measuring the running time at different resulting densities. We define the *density factor* $f = \frac{|E'|}{|\mathcal{T}||\mathcal{O}|}$, where $E' \subseteq E$ is the set of edges remaining after removals; the density factor of the original data, with edge set E, is 0.98.

Figs 9(a) and (b) plot the running times for finding top-10 patterns (with $\widehat{c} = 0.95$ and various ε), for the original $f = 0.98$ and other density factors, using unweighted and weighted cost functions, respectively. There are two separate observations here. First, the running time held fairly constant for $f \leq 0.9$. We believe this is due to two trends cancelling each other out: on the one hand, the size of S' increases linearly with decreasing f (due to higher densities in the complement), resulting in faster sorts; on the other hand, the number of edges in the complement that need to be scanned increases linearly with decreasing f. Second, the performance at $f = 0.98$ was significantly better than at smaller density factors. We profiled the code and found out most time of the algorithm is spent on sorting of the list $(\mathcal{O} - S')$ (see Algorithm 1). Our C++ implementation is based on *introsort*, which sorts a list much faster when almost all of the values are zeros; indeed, by far most of nodes in $\mathcal{O}$ were zero-degree in the complement graph at $f = 0.98$. We also varied $\widehat{c}$ from 0.95 to 0.8 and ran experiments with $\varepsilon = 0.1$ on the datasets of various density factors but the results did not really vary for different values of $\widehat{c}$.

V. Related Work

To the best of our knowledge, the problem in Definition 2.1 has not been studied before. However, there are two closely related problems of finding "dense" subgraphs (which is equivalent to satisfying the confidence threshold in our parlance) in bipartite graphs that have been studied. The first is where both sets ($\mathcal{T}$ and $\mathcal{O}$ in our notation) are totally ordered. This corresponds to summarizing a boolean matrix as a cover of dense rectangular submatrices, which was studied in [11]; the high dimensional version of this problem was studied in [2]. Here the subsets of nodes chosen from both sets are restricted to contiguous values (intervals). While this version of the problem has been shown to be NP-hard, polynomial-time heuristics (without guarantees) were given in these papers. The other related problem is where both sets are unordered. However, an instance of this is finding cliques in a bipartite graph, which has been shown to be NP-hard and inapproximable in polynomial time [10]. Heuristics for finding such "quasi-cliques" or "hyperrectangles" have been proposed in [1] and [15], respectively; the latter derives its results from association rules, itself an NP-hard problem. In particular, we share some of the framework, given the similar motivation, from [15] in this paper.

Data quality is the underlying motivation of this work. In particular, we are concerned with the (Cartesian) *completeness* of data. In databases, integrity constraints such as multivalued dependencies (MVDs), and similarly join dependencies, can be used to enforce such completeness. Given schema (A, B, C), MVDs assert that $\pi_{BC}(\sigma_{A=a}) = \pi_B(\sigma_{A=a}) \times \pi_C(\sigma_{A=a})$ for each unique a in $dom(A)$. While these constraints are different from the problem we study, we note that the discovery of MVDs has been studied in [16] and notions of approximate MVDs (that is, confidence measures) have been defined in [7]. The notion of a *pattern tableaux* has been used in the literature to summarize data quality with respect to *consistency*, using Conditional Functional Dependencies (CFDs) [8] and Conditional Inclusion Dependencies (CINDs) [9]. The optimization framework of these problems is similar to ours, where support and confidence thresholds are given and a parsimonious tableau satisfying the thresholds is discovered. In particular, we highlight so-called *range tableaux* described in [8], where any ordered attributes are summarized using an interval. We note that a tableau containing both a single ordered and a single unordered attribute is quite different from our problem in what it can express. The range tableau is only capable of expressing patterns combining an interval with either the wildcard (denoting all elements of the unordered set) or a singleton set (see Section IV for more details). In contrast, the sets discovered by our algorithm allow for a range on the ordered set to be combined with any arbitrary subset of the unordered set.

In addition to data quality, discovering dense regions of data is also a useful operation in data mining. For example, Optimized Support Rules are descriptive predicates between two ordered numerical attributes which hold to a high degree

[4]; in fact, the algorithm from [4] can be used to efficiently find a largest, as well as all maximal, submatrices having sufficient enough density (after which PARTIAL SET COVER can be run to find a smaller, non-redundant subset). Towards the same application, [5] discusses different ways to select and present such qualifying regions. Another example application is *burst detection* [17]. Co-clustering is a technique which reorders and partitions rows and columns of a matrix into homogenous hyperrectangles [13]. Some machine learning methods, such as decision trees, find and exploit dense regions in data to enable better prediction.

Finally, there is a relationship between graph summarization and compression. In particular, the discovery of dense bipartite cliques enables graphs to be represented more parsimoniously thus leading to good compression, as elaborated in [14].

VI. CONCLUSIONS

We have presented an approximation algorithm for summarizing missing (or corrupted) tuples in a relation over an ordered attribute (time) and an unordered attribute (object identifier) as a (hold or fail) *graph tableau*, since finding a smallest set of patterns is NP-hard. The algorithm efficiently finds a region of high (or low) density at each iteration and adds it to the tableau. We proved that the algorithm provides a $(2 + \ln |E|)$-approximation on the tableau size in $O(|\mathcal{T}|^2|\mathcal{O}|)$ time; and if we allow a $(1 + \epsilon)$-approximation on confidence, then the algorithm can be modified to run in $O(\frac{1}{\epsilon}|\mathcal{T}||\mathcal{O}| \log |\mathcal{T}|)$.

Future work includes extending the time intervals that are currently considered to richer temporal patterns such as "Monday afternoons in May from 14:00-17:00" and "the first weekday after a major holiday"; adding constraints on the way objects are grouped, for example, based on the spatial topology of the network; and streaming algorithms for online maintenance of tableaux over time.

VII. ACKNOWLEDGEMENTS

We thank Divesh Srivastava, Howard Karloff and Lukasz Golab for initial discussions that led to this work.

REFERENCES

[1] J. Abello, M. G. C. Resende, and S. Sudarsky. Massive quasi-clique detection. In *LATIN*, pages 598–612, 2002.

[2] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD Conference*, pages 94–105, 1998.

[3] G. Cormode, H. J. Karloff, and A. Wirth. Set cover algorithms for very large datasets. In *CIKM*, pages 479–488, 2010.

[4] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Data mining using two-dimensional optimized accociation rules: Scheme, algorithms, and visualization. In *SIGMOD Conference*, pages 13–23, 1996.

[5] B. J. Gao and M. Ester. Turning clusters into patterns: Rectangle-based discriminative data description. In *Proceedings of the Sixth International Conference on Data Mining (ICDM)*, pages 200–211, 2006.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[7] C. Giannella and E. L. Robertson. A note on approximation measures for multi-valued dependencies in relational databases. *Inf. Process. Lett.*, 85(3):153–158, 2003.

[8] L. Golab, H. J. Karloff, F. Korn, D. Srivastava, and B. Yu. On generating near-optimal tableaux for conditional functional dependencies. *PVLDB*, 1(1):376–390, 2008.

[9] L. Golab, F. Korn, and D. Srivastava. Efficient and effective analysis of data quality using pattern tableaux. *The Bulletin of IEEE Data Engineering*, 34(3):26–33, 2011.

[10] D. S. Hochbaum. Approximating clique and biclique problems. *J. Algorithms*, 29(1):174–200, 1998.

[11] L. V. S. Lakshmanan, R. T. Ng, C. X. Wang, X. Zhou, and T. Johnson. The generalized mdl approach for summarization. In *VLDB*, pages 766–777, 2002.

[12] E. L. Lawler. Fast approximation algorithms for knapsack problems. *Math. Oper. Res.*, 4(4):339–356, 1979.

[13] S. C. Madeira and A. L. Oliveira. Biclustering algorithms for biological data analysis: A survey. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 1(1):24–45, 2004.

[14] S. Navlakha, R. Rastogi, and N. Shrivastava. Graph summarization with bounded error. In *SIGMOD Conference*, pages 419–432, 2008.

[15] Y. Xiang, R. Jin, D. Fuhry, and F. F. Dragan. Succinct summarization of transactional databases: an overlapped hyperrectangle scheme. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 758–766, 2008.

[16] M. H. Yan and A. W.-C. Fu. Algorithm for discovering multivalued dependencies. In *CIKM*, pages 556–558, 2001.

[17] X. Zhang and D. Shasha. Better burst detection. In *ICDE*, page 146, 2006.

APPENDIX

A. Proof of Proposition 2.1

Proof: It is easy to see that the problem is in NP: given a tableau, a simple count of edges can verify confidence and support thresholds. To prove NP-hardness, we exhibit a polynomial-time reduction from BIN PACKING where the sizes are given in unary encoding, which is known to be strongly NP-complete [6]. Let V be the size of a bin and let $\{1, 2, ..., n\}$ be a collection of items to pack, each item j of size $a_j \leq V$; we assume V and the a_j's can be encoded in unary. We show how to use the Tableau Discovery Problem to partition these items such that they fit into the fewest number of bins.

First, we build a bipartite graph on vertex sets $\mathcal{T}$ and $\mathcal{O}$, where $\mathcal{T}$ is ordered and $\mathcal{O}$ contains an object corresponding to each item plus one additional object, as follows. Let $\mathcal{O} = \{o_1, ..., o_{n+1}\}$, where each o_j, for $j \in \{1, ..., n\}$, corresponds to the items of the same index, and let $\mathcal{T} = \{1, ..., 6nV\}$. Set $\hat{s} = 1$ and $\hat{c} = \frac{2}{3} + \frac{1}{6n}$. For each object $o_1, ..., o_n$, we encode the size of its corresponding item by creating edges to some $V - a_j$ nodes of $\mathcal{T}$ in the interval $[2nV + 1, 4nV]$. Specifically, we create edges $(4nV + j, j)$, $(4nV + n + j, j)$, ..., $(4nV + n(V - a_j) + j, j)$. Note that edges between nodes in $\mathcal{T}$ from $[2nV + 1, 4nV]$ and nodes $\{o_1, ..., o_n\}$ are so sparse that $conf(I, S) < \hat{c}$ for every subinterval $I \subseteq [2nV + 1, 4nV]$ and every subset $S \subseteq \{o_1, ..., o_n\}$. For each $o_j, j \in \{1, ..., n\}$, we also create an edge to each node of $\mathcal{T}$ in $[1, 2nV]$, as well to each node in $[4nV + 1, 6nV]$. The purpose of these edges is to make the confidence higher for nodes towards both ends of $\mathcal{T}$ so that the interval $[1, 6nV]$ will be selected over any smaller interval. Finally, from the special object o_{n+1} we insert edges to all nodes of $\mathcal{T}$ in $[1, 2nV]$, $[4nV + 1, 6nV]$ and $[2nV + \lfloor n/2 \rfloor V + 1, 2nV + \lceil n/2 \rceil V]$, a total of $4nV + 2V$ nodes.[7] Clearly, this construction can be done in polynomial time.

The special node o_{n+1} will need to be included in any set $S \subseteq \mathcal{O}$ in order for $conf(I, S) \geq \hat{c}$. Furthermore,

[7] If n is even, the range is $[2nV + (n/2)V + 1, 2nV + (n/2 + 1)V]$.

$I = [1, 6nV]$ will be chosen due to the insufficient degree of nodes in $[2nV + 1, 4nV]$ and high degree in $[1, 2nV]$ and $[4nV + 1, 6nV]$. The Hold Tableau Discovery algorithm, therefore, results in some smallest set (say, of size k) of pairs $([1, 6nV], S_1 \cup \{o_{n+1}\}), ..., ([1, 6nV], S_k \cup \{o_{n+1}\})$. The degree of o_{n+1} is $4nV + 2V$, which is large enough to satisfy the confidence threshold since $\frac{4nV+2V}{6nV} = \frac{2}{3} + \frac{2}{6n} > \hat{c}$. Therefore, o_{n+1} provides a budget of V for additional objects in each set. Without loss of generality, we can assume that the sets $S_1, ...S_k$ form a partition; otherwise, we can create a partitioned solution of the same size by setting $S'_1 = S_1$, $S'_2 = S_2 - S_1$, $S'_3 = S_3 - (S_1 \cup S_2)...$, $S'_k = S_k - (S_1 \cup \cdots \cup S_{k-1})$; note that $conf([1, 6nV], S'_\ell)$ must be at least as high as $conf([1, 6nV], S_\ell)$. Clearly, both $S_1, ..., S_k$ and $S'_1, ..., S'_k$ cover the same number of elements since $S_1 \cup \cdots \cup S_k = S'_1 \cup \cdots \cup S'_k$. The partition $S'_1, ..., S'_k$ also provides an optimal solution to BIN PACKING. ∎

B. Data Generator

Algorithm 2 shows the pseudo code of our data generator that generates the synthetic dataset we used in our experiments. The input of the generator includes $|\mathcal{T}|$ (as v_1), $|\mathcal{O}|$ (as v_2), the number of ground truth patterns k, the confidence threshold $\widehat{c}$, α for Zipf distributed random number generator, and β as the ratio of $\mathcal{T}$ nodes that will be used to create ground truth patterns. We use the $rand()$ function to generate a random number between 0 and 1 from uniform distribution, $\rho(P_i, P_{i+1}) = \frac{(|P_i|+|P_{i+1}|)(1-\widehat{c})}{\widehat{c}}$ computes the minimal distance between ground truth patterns $P_i = I_i \times S_i$ and $P_{i+1} = I_{i+1} \times S_j$, where the distance between P_i and P_j is equal to $I_{i+1}.lowerbound - I_i.upperbound$, assuming $I_{i+1}.lowerbound > I_i.upperbound$.

Algorithm 2 genSynData

Require: v_1, v_2, k, $\widehat{c}$, α, β
Ensure: synthetic dataset D
1: $s \leftarrow 0$
2: {Step 1: Calculate interval sizes and gap sizes}
3: initialize vector I of size k
4: **for** $i = 1 \rightarrow m$ **do**
5: $I[i] \leftarrow rand()$
6: $s \leftarrow s + I[i]$
7: initialize vector G of size $m - 1$
8: **for** $i = 1 \rightarrow k - 1$ **do**
9: $G[i] \leftarrow \max(rand() * 2, \rho(I[i], I[i + 1]))$
10: $s \leftarrow s + G[i]$
11: $sizeFactor \leftarrow v_1 * \beta / s$
12: {Step 2: Generate ground truth patterns}
13: $idx \leftarrow v_2 * (1 - \beta) * rand()$
14: $D \leftarrow \emptyset$
15: **for** $i = 1 \rightarrow k$ **do**
16: $iSize \leftarrow I[i] * sizeFactor$
17: $b_2 \leftarrow Zipf(v_2, \alpha)$
18: $ranV2 \leftarrow$ randomly pick b_2 girls for the pattern
19: $b_e \leftarrow b_2 * iSize * \widehat{c}$
20: $E \leftarrow$ randomly pick b_e edges of sub bipartite graph $(idx, idx + iSize) \times ranV2$
21: $D \leftarrow D \cup E$
22: $idx \leftarrow idx + iSize + G[i] * sizeFactor$
23: **return** D