Sequential Dependencies

Lukasz Golab
AT&T Labs–Research
lgolab@research.att.com

Howard Karloff
AT&T Labs–Research
howard@research.att.com

Flip Korn
AT&T Labs–Research
flip@research.att.com

Avishek Saha
University of Utah
avishek@cs.utah.edu

Divesh Srivastava
AT&T Labs–Research
divesh@research.att.com

ABSTRACT

We study *sequential dependencies* that express the semantics of data with ordered domains and help identify quality problems with such data. Given an interval g, we write $X \rightarrow_g Y$ to denote that the difference between the Y-attribute values of any two consecutive records, when sorted on X, must be in g. For example, $time \rightarrow_{(0,\infty)} sequence\_number$ indicates that sequence numbers are strictly increasing over time, whereas $sequence\_number \rightarrow_{[4,5]} time$ means that the time "gaps" between consecutive sequence numbers are between 4 and 5. Sequential dependencies express relationships between ordered attributes, and identify missing (gaps too large), extraneous (gaps too small) and out-of-order data.

To make sequential dependencies applicable to real-world data, we relax their requirements and allow them to hold approximately (with some exceptions) and conditionally (on various subsets of the data). This paper proposes the notion of conditional approximate sequential dependencies and provides an efficient framework for discovering pattern tableaux, which are compact representations of the subsets of the data (i.e., ranges of values of the ordered attributes) that satisfy the underlying dependency. We present analyses of our proposed algorithms, and experiments on real data demonstrating the efficiency and utility of our framework.

1. INTRODUCTION

Interesting data sets often contain attributes with ordered domains: timestamps, sequence numbers, surrogate keys, measured values such as sales, temperature and stock prices, etc. Understanding the semantics of such data is an important practical problem, both for data quality assessment as well as knowledge discovery. However, integrity constraints such as functional and inclusion dependencies do not express any ordering properties. In this paper, we study *sequential dependencies* for ordered data and present a framework for discovering which subsets of the data obey a given sequential dependency.

Given an interval g, a sequential dependency (SD) on attributes X and Y, written as $X \rightarrow_g Y$, denotes that the distance between the Y-values of any two consecutive records, when sorted on X, are within g. SDs of the form $X \rightarrow_{(0,\infty)} Y$ and $X \rightarrow_{(-\infty,0]} Y$ specify that Y is strictly increasing and non-increasing, respectively, with X, and correspond to classical Order Dependencies (ODs) [11]. They are useful in data quality analysis (e.g., sequence numbers must be increasing over time) and data mining (in a business database, delivery date increases with shipping date, in a sensor network, battery voltage increases with temperature, etc.) SDs generalize ODs and can express other interesting relationships between ordered attributes. An SD of the form $sequence\_number \rightarrow_{[4,5]} time$ specifies that the time "gaps" between consecutive sequence numbers are between 4 and 5. In the context of data quality, SDs can measure the quality of service of a data feed that is expected to arrive with some frequency, e.g., a stock ticker that should generate updated stock prices every 4 to 5 minutes. In terms of data mining, the SD $date \rightarrow_{[20,\infty)} price$ identifies stock prices that rapidly increase from day to day (by at least 20 points). Related examples may be found in [11, 23].

In practice, even "clean" data may contain outliers. We characterize the degree of satisfaction of an SD by a given data set via a *confidence* measure. Furthermore, real data sets, especially those with ordered attributes, are inherently heterogeneous, e.g., the frequency of a data feed varies with time of day, measure attributes fluctuate over time, etc. Thus, we consider *Conditional Sequential Dependencies* (CSDs), which extend SDs analogously to how Conditional Functional Dependencies extend traditional FDs [4]. A CSD consists of an underlying SD plus a representation of the subsets of the data that satisfy this SD. Similar to CFDs, the representation we use is a *tableau*, but here the tableau rows are intervals on the ordered attributes.

1.1 Network Monitoring Examples

Internet Service Providers (ISPs) collect various network performance statistics, such as the number of packets flowing on each link. These measurements are maintained by routers in the form of cumulative counters, which are probed periodically by a data collection system. A plot of packet counts versus time is shown in Figure 1. While we expect the counts to increase over time, counters are finite (e.g., 32 bits) and thus periodically loop around. Furthermore, counters reset whenever the router is rebooted. Additionally, spurious measurements may appear (e.g., at time 16 in Figure 1), such as when the data collector probes the wrong router. Due to the cyclic nature of the counters, the semantics of this data set cannot be captured by the SD $time \rightarrow_{(0,\infty)} count$; we need a *conditional* SD whose tableau identifies subsets that satisfy the embedded SD. For instance, each pattern in Tableau A from Figure 1 corresponds to an interval that exactly satisfies the embedded SD. Alternatively, we may allow a small number of violations in order to produce more informative tableaux and help avoid "overfitting" the data. Tableau B from Figure 1 contains two patterns that cap-

VLDB '09, August 24-28, 2009, Lyon, France

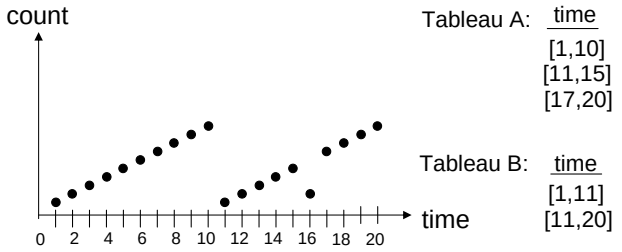

Figure 1: Tableaux for an SD $time \rightarrow_{(0,\infty)} count$.

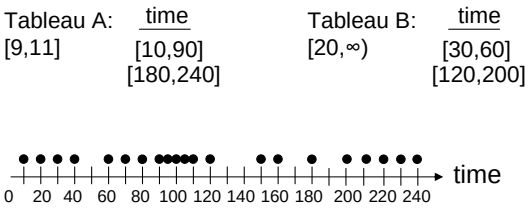

Figure 2: Tableaux for an SD $pollnum \rightarrow_r time$.

ture the two mostly-increasing fragments of the data set (with one violation at time 16). It not only identifies the intervals over which the SD is obeyed but also pinpoints the time at which there is a disruption in the ordering (at time 11). Such tableaux are useful tools for concisely summarizing the data semantics and identifying possible problems with the network or the data collector, e.g., a tableau with many "short" patterns suggests premature counter roll-over.

An ISP may also be interested in auditing the polling frequency. The data collector may be configured to probe the counters every ten seconds; more frequent polls may indicate problems at the collector (it may be polling the same router multiple times) while missing data may be caused by a misconfigured collector or a router that is not responding to probes. A possible sequence of measurement times (not the actual counter values) is shown in Figure 2, sorted in polled order, along with a tableau (labeled Tableau A) for the embedded SD $pollnum \rightarrow_{[9,11]} time$, which asserts that the gaps between adjacent polls should be between 9 and 11 seconds. Again, we allow each pattern to contain a small number of violations to better capture the trends in the data; e.g., the first pattern $[10, 90]$ contains one gap of length 20.

Furthermore, testing related SDs with different gap ranges reveals intervals that violate the expected semantics. For example, $pollnum \rightarrow_{[20,\infty)} time$ finds subsequences with (mostly) long gaps, as shown in Tableau B. Similarly, $pollnum \rightarrow_{[0,10)} time$ detects periods of excessively frequent measurements. The corresponding tableaux provide concise representations of subsets that deviate from the expected semantics, and are easier to analyze by a user than a raw (possibly very lengthy) list of all pairs of records with incorrect gaps. It is worth noting that simply counting the number of polls to detect problems is insufficient: if the window size for counts is too small (say, ten seconds), then false positives can occur if polls arrive slightly late; if the window size is too large (say, one hour), then false negatives can occur due to missing and extraneous data "canceling each other out".

1.2 Contributions

Our main contribution is a novel integrity constraint for ordered data. The mechanisms generating ordered data often provide the order semantics—sequence numbers are increasing, measurements arrive every ten seconds, etc. However, finding subsets of the data obeying the expected semantics is laborious to do manually. We therefore assume that the embedded SD has been supplied and solve the problem of discovering a "good" pattern tableau. Following the criteria set forth in [13], we desire *parsimonious* tableaux that use the fewest possible patterns to identify a large fraction of the data ("support") that satisfy the embedded SD with few violations ("confidence"). Our technical contribution is a general framework for CSD tableau discovery, which involves generating candidate intervals and constructing a tableau using a smallest subset of candidate intervals (each of which has sufficiently high confidence) that collectively "cover" the desired fraction of the data.

In our model, every tableau pattern must independently satisfy the embedded SD. The brute force algorithm computes the confidence of all $\Theta(N^2)$ possible intervals (in a sequence of N elements) and identifies as candidates those which have a sufficiently high confidence. Since our goal is to discover a concise tableau, we prefer large intervals that cover more data, and therefore can ignore candidate intervals contained in larger candidate intervals. Our first observation is that CSDs obey a "prefix" property, whereby the confidences of all prefixes of a given interval I are incrementally computed en route to computing the confidence of I itself. Thus, it suffices to compute the confidence of the N intervals $[i, N]$, where $1 \leq i \leq N$, and, for each i, find the maximum j such that the interval $[i, j]$ has the required confidence.

Our second observation is that CSDs also satisfy a "containment" property, which implies that the confidence of an interval slightly larger than some interval I must be similar to that of I. We give an approximation algorithm that computes the confidence of a small set of carefully chosen intervals such that, for each candidate interval I identified by the exact algorithm, our algorithm is guaranteed to identify a slightly larger interval whose confidence is not much lower than that of I. Instead of computing the confidence of the N intervals described above, the approximation algorithm only needs to compute the confidence of $O((\log N)/\delta)$ intervals, where $1 + \delta$ is a bound on the approximation error.

In addition to improving the efficiency of the candidate generation phase, our framework improves the efficiency of the tableau construction step. This step solves the partial interval cover problem by choosing the fewest candidate intervals that cover the desired fraction of the data. An exact dynamic programming algorithm for this problem takes quadratic time in the number of candidate intervals. We give a linear-time and -space greedy heuristic and prove that it returns tableaux with sizes within a constant factor (of nine) of the optimal solution.

To summarize, our overall contributions are as follows.

- We propose Conditional Sequential Dependencies as novel integrity constraints for ordered data and give efficient algorithms for testing their confidence.
- We give a general framework that makes the discovery of "good" tableaux for CSDs computationally feasible, provided that the confidence measure satisfies the prefix and containment properties.
- We present experimental results demonstrating the efficiency (order-of-magniture improvement over the brute force approach), as well as the utility (in revealing useful data semantics and data quality problems), of the proposed framework on a wide range of real data sets.

The rest of the paper is organized as follows. Section 2 defines the problems we study. Section 3 presents our framework. Section 4 summarizes our experimental results. Section 5 compares our contributions with related work. Section 6 concludes and suggests areas for future work.

2. DEFINITIONS

Let S be a relational schema on attributes $A_1, A_2, ..., A_k$ with relation instance $R = \{t_1, t_2, ..., t_N\}$. Let $dom(X) = \{t_1[X], t_2[X], ..., t_N[X]\}$ refer to the set of domain values over X, where $t[X]$ denotes the relation tuple t projected on the attributes X. We model the input to our problem as a relation, some of whose attributes have ordered domains.

DEFINITION 1 *Let X and Y, $X \subseteq S$ and $Y \subseteq S$, be two attribute sets, g be an interval, and π be the permutation of rows of R increasing on X (that is, $t_{\pi(1)}[X] < t_{\pi(2)}[X] < \cdots < t_{\pi(N)}[X]$). A* sequential dependency *(SD) $X \rightarrow_g Y$ is said to hold over R if for all i such that $1 \leq i \leq N-1$, $t_{\pi(i+1)}[Y] - t_{\pi(i)}[Y] \in g$. That is, when sorted on X, the gaps between any two consecutive Y-values must be within g.*

We refer to X as the antecedent of the SD and Y as the consequent. We assume that total orderings exist on X and Y, and that there is a mapping $f()$ which linearizes the different combinations of attribute values in X and Y into integers. For example, if $X = \{hour, minute, second\}$ then the tuple $t[X] = (h, m, s)$ could be mapped via $f(h, m, s) = 3600h + 60m + s$.

2.1 Approximate Sequential Dependencies

In practice, an SD may not hold exactly: when ordered on X, the resulting sequence of Y-values may not have the correct gaps. Previous work on characterizing the extent to which ODs [8], FDs [19] and CFDs [13] hold on a given relation instance employed a "deletion-based" metric that determines the largest possible subset of the relation that satisfies the constraint. Using this measure, the *confidence* of interval $[11, 20]$ from Figure 1 w.r.t. the SD $time \rightarrow_{(0,\infty)} count$ is $\frac{9}{10}$ since the largest subset that makes the SD valid contains every record except the one at time 16. The confidence of the entire data set, i.e., the interval $[1, 20]$, is $\frac{10}{20}$ since the largest satisfying subset contains (the first) ten points.

Now consider the interval $[10, 90]$ from Figure 2. To satisfy the SD $pollnum \rightarrow_{[9,11]} time$, we select either the first four points or the last four points in this interval, for a confidence of $\frac{4}{8}$. This confidence value seems low since this interval has only one "problem", namely a missing record around time 50. We thus define the confidence of a CSD using the following edit distance metric, which is a natural extension of the well-known deletion metric.

DEFINITION 2 *The* confidence *of a suggested SD over a given relation instance (or subset thereof) of size N is $\frac{N-OPS}{N}$, where OPS is the smallest possible number of records that need to be inserted or deleted to make the SD hold.*

Note that confidence cannot be negative since in the worst case, we can "delete" all but one record, which will trivially satisfy the SD. This metric has several useful properties. It is robust to occasional missing data—in the above example, the interval $[10, 90]$ has a confidence of $\frac{7}{8}$ since only one edit operation (insertion) needs to be made to satisfy the SD. It is also robust to spurious values. Returning to the above example, the sequence $< 10, 20, 30, 1000, 40 >$ has a relatively high confidence of $\frac{4}{5}$ since it suffices to delete the suspicious element 1000. Furthermore, our metric penalizes based on gap sizes, unlike just counting the fraction of "bad gaps" (i.e., those not in the specified gap range). For example, if one is expecting all gaps to be between 3 and 5, then a gap of 6 can be corrected by one insertion, but a gap of size 1000 requires 199 insert operations. We will point out several other metrics that are compatible with our tableau generation framework in Section 3.3.

2.2 Computing the Confidence of SDs

Having defined the confidence of a SD, we now describe how to efficiently compute it (i.e., compute OPS) on a relation instance.

2.2.1 Confidence of Simple SDs

First, consider a "simple" SD of the form $X \rightarrow_{(0,\infty)} Y$, which requires Y to be increasing with X. Note that this SD does not limit the maximum gap length, so we need not insert new records to reduce the lengths of oversized gaps. Its confidence may be computed from the length of the *longest increasing subsequence* on Y, after ordering the relation on X. More formally, let π be the permutation of rows of R increasing on X. We wish to find a longest subsequence $\pi(i_1) < \pi(i_2) < \cdots < \pi(i_T)$ of π, $i_1 < i_2 < \cdots < i_T$, such that $t_{\pi(i_1)}[Y] < \cdots < t_{\pi(i_T)}[Y]$, for some $T \leq N$. Let S_N be the sequence $< t_{\pi(1)}[Y], \ldots, t_{\pi(N)}[Y] >$. We denote the *length* (not the subsequence itself) of the longest increasing subsequence of S_N by $LIS(S_N)$. Then the confidence of an SD on R is $LIS(S_N)/N$, which can be computed in $O(N \log N)$ time [10]. In general, SDs of the form $X \rightarrow_{[G,\infty)} Y$, G a finite non-negative integer, can be handled in a similar way, by finding longest sequences increasing by at least G at every step. We note that other measures of "sortedness" may be natural for some applications (such as based on number of inversions [16, 14], average inversion length or "satisfaction within bounds"[8]) and could be used in place of this quantity throughout this paper and can be computed within the same time complexity by our framework.

2.2.2 Confidence of Other SDs

We now consider general SDs of the form $X \rightarrow_{[G_1,G_2]} Y$, where $0 \leq G_1 \leq G_2 \neq 0$. We say that a sequence (of Y-values mapped to integers, when sorted on X) is *valid* if it is non-empty, all elements are integers, and all its gaps are between G_1 and G_2. Computing the confidence requires finding $OPS(N)$—the minimum number of integers that must be added to or deleted from the length-N sequence in order to obtain a valid sequence. For example, the confidence of an SD with $G_1 = 4$ and $G_2 = 6$ on the sequence $< 5, 9, 12, 25, 31, 30, 34, 40 >$ is $1 - 4/8 = 1/2$: deleting 12 and inserting 15 and 20 in its place (or deleting 5, 9 and 12) and then deleting 31 will convert the sequence into a valid one, and no series of three or fewer insertions and deletions will make the sequence valid. In general, the sequence need not be sorted, i.e., some gaps may be negative.

Given a sequence $< a_1, a_2, ..., a_N >$ of integers, for $i = 1, 2, ..., N$ let $v = a_i$ and define $T(i)$ to be the minimum number of insertions and deletions one must make to $< a_1, a_2, ..., a_i >$ in order to convert it into a valid sequence ending in the number v. (Note that since the value v might appear more than once in the sequence, one might get a sequence ending in a copy of v which is not the ith element in the sequence.) Now computing $OPS(N)$ from the $T(i)$'s can be done as follows: $OPS(N) = \min_{0 \leq r \leq N-1}\{r + T(N-r)\}$, as proven in Claim 3.

CLAIM 3 *The minimum number $OPS(i)$ of insertions and deletions required to convert an input sequence S_i into a valid one is given by $\min_{0 \leq r \leq i-1}\{r + T(i-r)\}$. Furthermore, $OPS(i)$ can be calculated inductively by $OPS(1) = 0$ and $OPS(i) = \min\{1 + OPS(i-1), T(i)\}$ for all $i \geq 2$.*

PROOF. We first prove that $OPS(i) \geq \min_{0 \leq r \leq i-1}\{r + T(i-r)\}$. In the optimal transformation, let r be the exact number of terms at the end of the sequence $S_i =< a_1, a_2, ..., a_i >$ which are removed; hence, a_{i-r} remains and appears in the final sequence. Clearly, $0 \leq r \leq i-1$. After removing those r terms, the optimal algorithm must transform the prefix consisting of the first $i - r$

terms into a valid sequence ending in a_{i-r}. The cost to do this is $T(i-r)$, and hence the optimal total cost is $r + T(i-r)$. Since there is some r, $0 \le r \le i-1$, such that $OPS(i) = r + T(i-r)$, we can infer that $OPS(i) \ge \min_{0 \le r \le i-1}\{r + T(i-r)\}$.

Clearly $OPS(i) \le \min_{0 \le r \le i-1}\{r + T(i-r)\}$ as well, since for each such r one could get a valid sequence by deleting the last r integers and then, at cost $T(i-r)$, converting the sequence $< a_1, a_2, ..., a_{i-r} >$ into a valid sequence ending in the value a_{i-r}.

The second statement follows from $OPS(i) = \min_{0 \le r \le i-1}\{r + T(i-r)\}$ by splitting off the $r = 0$ case from the $1 \le r \le i-1$ case. □

In order to show how to compute the $T(i)$'s, we need a definition of and a lemma about $dcost$, a function which specifies the fewest integers one must append to a length-1 sequence to get a valid sequence whose last element is exactly d larger than its first.

DEFINITION 4 *Define $dcost(d)$, for $d = 0, 1, 2, ...$, to be the minimum number of integers one must append to the length-1 sequence $< 0 >$ to get a valid sequence ending in d, and ∞ if no such sequence exists.*

It is nontrivial but not hard to prove the following lemma, whose proof we omit due to space constraints.

LEMMA 5 *If $G_1 = 0$, then $dcost(d) = \lceil d/G_2 \rceil$. Otherwise, $dcost(d) = \lceil d/G_2 \rceil$ if $\lceil (d+1)/G_1 \rceil > \lceil d/G_2 \rceil$ and ∞ otherwise.*

For example, if $G_1 = 4$ and $G_2 = 6$, then $dcost(7) = \infty$. Furthermore, $dcost(8) = 2$, uniquely obtained with two gaps of length 4. This is interesting since one might be tempted to infer from "$dcost(d) = \lceil d/G_2 \rceil$" that all but one gap have length G_2.

LEMMA 6 *Choose an i, $1 \le i \le N$. Let $v = a_i$. Then among all ways to convert $< a_1, a_2, ..., a_i >$ into a valid sequence ending in the number v, there is one in which the ith symbol is never deleted.*

Keep in mind that $v = a_i$ may appear more than once in the sequence $< a_1, a_2, ..., a_i >$. If one generates a valid sequence ending in the value v, just which v is it? The v which is the ith symbol in the sequence? Or the v which is the jth, for some $j < i$ with $a_j = a_i = v$? The content of this lemma is that there is always a minimum-cost way of transforming the sequence into a valid sequence in which v is the ith symbol, not the jth.

PROOF. If the ith symbol is deleted, let j be the largest index of a nondeleted symbol (which must exist). Clearly $a_j \le a_i$, since in the final list all integers are at most $v = a_i$. If $a_j < a_i$, then the algorithm must at some point append an a_i, but then it was wasteful to delete the ith integer in the first place, and so it should not have. Hence we may assume that $a_j = a_i$. Now instead of deleting the ith symbol and not deleting the jth, delete the jth and do not delete the ith. □

THEOREM 7 *Suppose that we have already computed $T(1), T(2), ..., T(i-1)$, for some $i \le N$. We can compute $T(i)$ using the existing $T(1), ..., T(i-1)$ as follows. Define*

$$min_1 := i - 1,$$

$$min_2 := \min_{j:j<i, a_j < a_i} \{T(j) + (i-1-j) + [dcost(a_i - a_j) - 1]\},$$

and define

$$min_3 := \min_{j:j<i, a_j = a_i} \{T(j) + (i-1-j)\}.$$

Then $T(i) = \min\{min_1, min_2\}$ if $G_1 > 0$ and $T(i) = \min\{min_1, min_2, min_3\}$ if $G_1 = 0$.

PROOF. Choose i, let $v = a_i$, and consider an optimal sequence of moves which converts $< a_1, a_2, ..., a_i >$ into a valid sequence whose last entry is v. By Lemma 6, we may assume that the optimal sequence of moves does not delete the ith entry. Either the optimal sequence deletes the first $i-1$ integers or it does not. If it does, its cost is obviously $i-1$. If it does not, then let j be the maximum index less than i such that the jth symbol is not deleted. Clearly $a_{j+1}, a_{j+2}, ..., a_{i-1}$, a total of $i-1-j$ integers, are deleted.

If $G_1 > 0$, then, since a_i is not deleted, $a_j < a_i$. The adversary, who converts the input sequence into a valid sequence using the fewest operations, will then "bridge the gap" from a_j to a_i, and convert $< a_1, ..., a_j >$ into a valid sequence ending at a_j, at a cost of $T(j)$. Given a length-2 integral sequence $< y, z >$, $y \le z$, the number of integers one must insert between y and z to get a valid sequence (i.e., to "bridge the gap" from y to z) is

- 0 if $y = z$ and $G_1 = 0$,
- ∞ if $y = z$ and $G_1 > 0$, and
- $dcost(z - y) - 1$ if $y < z$.

Hence, the total cost is $(i-1-j) + (dcost(a_i - a_j) - 1) + T(j)$.

If $G_1 = 0$, there is the additional possibility that $a_j = a_i$. The cost of bridging the gap is zero, for a total cost of $(i-1-j) + T(j)$. □

Now that we have a recurrence for computing the $T(i)$'s, we discuss how one can use the recurrence to calculate all the $T(i)$'s quickly. If, for each a_i, every a_j-value with $j < i$ is evaluated for the recurrence, then the algorithm will run in linear time for each i, or quadratic time in total. However, it is possible, for each i, to find the best j without trying all the linearly-many j's. The idea here is that the $dcost$ values are either finite or infinite. Clearly any term having an infinite $dcost$ can be ignored. The observation is that the infinite $dcost$s come in a limited number of consecutive blocks, and hence the finite $dcost$s also come in a limited number of consecutive blocks (all but one of which have finite size), which we call *bands*. We will show how to quickly compute the minimum over one band, and therefore, for each i, the time to compute a_i will be bounded by the product of the number of bands and the time per band. The overall time will be just N times this product.

Given some gap range $[G_1, G_2]$, the bands of finite values for $dcost$ are the input value ranges $[kG_1, kG_2]$, for integers $k \ge 1$. Note that these bands widen with increasing k (if $G_1 < G_2$). Indeed, when k becomes large enough, the bands will overlap and, therefore, no more $dcost$ values of ∞ will occur for d this large. Exactly when will the overlap first occur? There is no space for a d with $dcost(d) = \infty$ between the band $[\ell G_1, \ell G_2]$ and the next band $[(\ell+1)G_1, (\ell+1)G_2]$ if and only if $(\ell+1)G_1 \le \ell G_2 + 1$, i.e., $\ell \ge \lceil (G_1 - 1)/(G_2 - G_1) \rceil$ (if $G_1 \ne G_2$). We shall deal with the case where $G_1 = G_2$ separately below.

Given a fixed a_i, the formula for $T(i)$ requires that we compute $dcost(a_i - a_j)$; hence, we wish to find the values of a_j for which $dcost(a_i - a_j)$ is finite. Since $dcost(d)$ is finite within bands $kG_1 \le d \le kG_2$ for each k, substituting $d = a_i - a_j$ and solving for a_j yields bands $a_i - kG_2 \le a_j \le a_i - kG_1$. So the bands with respect to a_j are now $[a_i - G_2, a_i - G_1], [a_i - 2G_2, a_i - 2G_1], ..., [a_i - (\ell-1)G_2, a_i - (\ell-1)G_1]$ and one band of infinite length $[-\infty, a_i - \ell G_1]$. Since the a_j's come from sequence element values, clearly we never need to consider a_j-values less than the smallest value a_{min} in the sequence. Thus, we can threshold any band extending below a_{min}, ensuring that no band is of infinite length (i.e., if a_{min} lies within $[-\infty, a_i - \ell G_1]$ then this band gets

truncated to $[a_{min}, a_i - \ell G_1]$) and possibly resulting in fewer than ℓ bands to search. Note that, since in each of these bands $dcost$ is finite, $dcost(d)$ is equivalently defined as $\lceil d/G_2 \rceil$. Furthermore, since $0 \leq \lceil x \rceil - x < 1$ for all x, we can substitute the function d/G_2 in place of $\lceil d/G_2 \rceil$ and obtain the same result, because all the other variables are integers so adding a fractional amount less than 1 will not change the rank order for the best a_j.

Here is how the algorithm proceeds. For a fixed i, in any band (with finite $dcost$) $\arg\min_{j:j<i, a_j<a_i}\{T(j) + (i-1-j) + [dcost(a_i - a_j) - 1]\}$ is equivalent to $\arg\min_{j:j<i, a_j<a_i}\{T(j) - j - a_j/G_2\}$. So for each band k ($1 \leq k \leq \ell$), we find $j^{(k)} = \arg\min_j\{T(j) - j - a_j/G_2\}$ subject to $a_j \in [a_i - kG_2, a_i - kG_1]$, or subject to $a_j \in [a_{min}, a_i - kG_1]$ if $a_i - kG_2 < a_{min}$. Let j^* be the minimum j from among these bands, that is, $j^* = \min_k\{j^{(k)}\}$. Then $min_2 = T(j^*) + (i - 1 - j^*) + [dcost(a_i - a_{j^*}) - 1]$. We also need to consider the j's for which $a_j = a_i$. So we let $j' = \arg\min_{j:j<i, a_j=a_i}\{T(j) - j - a_j/G_2\}$ and $min_3 = T(j') + (i - 1 - j')$. Finally, we take $T(i) = min\{min_1, min_2\}$ if $G_1 > 0$ and $T(i) = \min\{min_1, min_2, min_3\}$ if $G_1 = 0$.

For the case of $G_1 = G_2 = G$, given some integer $G > 0$, the algorithm is simpler and can be computed in $O(N \log N)$ time. The idea is to partition the sequence elements a_j into G classes $0, 1, ..., G-1$ based on their $(\mathrm{mod} G)$-values. Then, given a_i, we search only the a_j's whose $a_j = a_i (\mathrm{mod} G), a_j \leq a_i$, and take the j with smallest $T(j) - j - a_j/G$ as j^*. Clearly, j^* can be found in $O(\log N)$ time. As usual, we let $min_2 = T(j^*) + (i - 1 - j^*) + [dcost(a_i - a_{j^*}) - 1]$.

THEOREM 8 *The confidence of an SD $X \rightarrow_{[G_1,G_2]} Y$ on a sequence of length N can be computed in time $O(\frac{G_2}{G_2 - G_1} N \log N)$ when $G_1 \neq G_2$ and in time $O(N \log N)$ when $G_1 = G_2$.*

PROOF. For each of N sequence elements, we search in at most $\frac{G_1 - 1}{G_2 - G_1} + 1 = \frac{G_2}{G_2 - G_1}$ bands for the $\arg\min$, and each band can be searched and updated in $O(\log N)$ time using a standard data structure for range-$\min$ over arbitrary ranges of values. In fact, we can afford to first sort the sequence element values, thus transforming them into their ranks, and store the $\min$ over each dyadic interval in rank-space. That way, the ranges can be transformed into being over a universe of size N (i.e., the ranks) – which makes updates much easier – and a range-$\min$ can be stored for every possible binary partition of the values with respect to their ranks. Then range query intervals can be decomposed into $O(\log N)$ adjacent dyadic intervals, from which the result can be obtained. The total query time is the product of these, $O(\frac{G_2}{G_2 - G_1} N \log N)$. □

2.3 Conditional Dependencies

DEFINITION 9 *A* Conditional Sequential Dependency *(CSD) is a pair $\phi = (X \rightarrow_g Y, T_r)$, where $X \rightarrow_g Y$, referred to as the* embedded *SD, and T_r is a "range pattern tableau" which defines over which rows of R the dependency applies. Each pattern $t_r \in T_r$ specifies a range of values of X that identify a subset of R (subsequence on X). The CSD states that, for each $t_r \in T_r$, the embedded SD independently holds over the subset of the relation (subsequence on X) identified by t_r.*

Let $[T_{\pi(i)}[X], T_{\pi(j)}[X]]$ be the interval represented by a tableau pattern t_r; again, we let π be the permutation of rows in R sorted on X. We define the *confidence* of t_r as the confidence of its interval w.r.t. the embedded SD, the *support* of t_r as the number of records contained in its interval, i.e., $j - i + 1$, and the *position interval* of t_r as $[i, j]$ (for example, the position interval of the pattern $[30, 60]$ from Tableau B in Figure 2 is $[3, 5]$). We also define the *total support*, or *global support*, of a CSD as the support of the union of the intervals identified by the tableau patterns (note that patterns may overlap).

2.4 Tableau Discovery Problem

The goal of tableau discovery is to find a parsimonious tableau whose patterns all provide sufficient confidence and describe a sufficient portion of the data. Thus, given a relation instance and an embedded SD, we wish to find a smallest tableau (if any exists) subject to confidence and (global) support threshold constraints.

DEFINITION 10 *The* CSD Tableau Discovery Problem *is, given a relation instance R, an embedded SD $X \rightarrow_g Y$, a global support threshold $\hat{s}$ and a confidence threshold $\hat{c}$, to find a tableau T_r of minimum size such that the CSD $\phi = (X \rightarrow_g Y, T_r)$ has a global support at least $\hat{s}$ and that each $t_r \in T$ has confidence at least $\hat{c}$.*

Naturally, one could optionally impose a local support threshold that is met by each tableau pattern, in order to ensure that spurious and uninteresting patterns are not reported. Furthermore, rather than seeking a tableau with a sufficiently high global support, it may be useful to ask for the k "best" patterns (e.g., those having the highest local support) regardless of the global support.

3. TABLEAU DISCOVERY

Here we propose a general tableau discovery framework. We assume that the confidence of an interval I containing N points may be written as $\frac{f(I)}{N}$, where f is some aggregate function, and that $0 \leq f(I) \leq N$ to ensure that confidence is between zero and one. For our confidence metric, $f(I) = N - OPS$ and $1 \leq f(I) \leq N$ since we never need to make more than $N-1$ edit operations (recall Section 2.1). Our framework consists of two phases: (1) generating candidate intervals and (2) choosing from these candidates a small subset providing suitable (global) support to be used for the tableau. What makes the first phase inherently challenging is that the confidence of an interval may not be readily composed from those of its subintervals due to the complex nature of whatever aggregate function is employed in measuring confidence. Take Figure 1 for example. The confidence of the interval $[1, 10]$ is 1, the confidence of $[11, 20]$ is 0.9, but the confidence of $[1, 20]$ is only 0.5. However, the following properties can be exploited.

DEFINITION 11 *An aggregate function f over a sequence is said to satisfy the* prefix property *if the time to compute f on all prefixes of a sequence is no more than a constant greater than the time to compute it on the sequence itself. Hence the prefix property is a property of the* algorithm *computing f, rather than f itself. Formally, we are given some time bound $g(N)$ and we need to assume that the property can be computed on all N prefixes of a sequence of length N in time $g(N)$, in total.*

DEFINITION 12 *An aggregate function f is said to satisfy the* containment property *if for any sequence σ and subsequence τ appearing in* consecutive *positions of σ, $f(\tau) \leq f(\sigma)$.*

First we show that our framework can be used to speed up interval generation with any confidence measure whose aggregate function f obeys both the prefix property and the containment property. Our emphasis will be on developing scalable algorithms (i.e., running in time $N\mathrm{polylog}N$). We then illustrate the framework using the confidence measure from Definition 2, followed by a discussion of other applicable metrics.

3.1 Generating Candidate Intervals

Only intervals satisfying the supplied confidence threshold are considered as tableau candidates. Given a choice between any two candidates, where one is contained in the other, choosing the smaller one may unnecessarily increase the size of the tableau. Hence, for each i, we want max $j \geq i$ (if any) such that the position interval $[i, j]$ has confidence at least $\hat{c}$ (in the remainder of this section, we will refer to position intervals as intervals unless otherwise noted). There are at most N such intervals as there is at most one with each given left endpoint. (One could go further and remove all intervals contained in others.)

A naive way to find candidate intervals would be to compute the confidence of all $N(N+1)/2$ possible intervals between 1 and N. Using the prefix property this can be improved by a factor of N by computing confidence over the intervals $[1..N], [2..N], \ldots, [N-1..N]$ and using intermediate results. Unfortunately, this is still too expensive for large data sets if computing the confidence on an interval of length ℓ requires $\Omega(\ell)$ time, as it will require $\Omega(N^2)$ time to find all maximal intervals. How can we find these intervals without testing all (i, j) pairs? The trick, at the price of "cheating" on the confidence (as described below), is to test only a proper subset of the pairs, but enough so that, for any interval I chosen by an adversary (i.e., any interval which could appear in an optimal tableau), our set of candidate intervals contains one, J, which contains I and whose length is only slightly larger, specifically, $|J| \leq (1+\epsilon)|I|$. Any aggregate function f satisfying the containment property will satisfy $f(J) \geq f(I)$, and hence its confidence $f(J)/|J|$ will be at least $f(I)/|J| \geq f(I)/[(1+\epsilon)|I|] = (f(I)/|I|)/(1+\epsilon)$, and hence at least $1/(1+\epsilon)$ times as large as I's. Thus, by "cheating" on confidence (but only by the small factor $1/(1+\epsilon)$), we can ensure that every adversary interval is (barely) covered by some candidate interval.

We give an approximation algorithm for efficiently generating candidate intervals. The algorithm takes a real $\epsilon > 0$ and builds a set of *real* intervals in $[0, N]$, with the following property. For any subinterval I of $[0, N]$ of length at least 1, among the intervals generated by the algorithm is an interval J which contains I and whose length is at most $1+\epsilon$ times as large.[1]

Now we generate the intervals. Let us choose a small positive δ whose value will be determined later. For each length of the form $\ell_h = (1+\delta)^h$, for $h = 0, 1, 2, ...$, until $(1+\delta)^h$ first equals or exceeds N, build a family of intervals each of length ℓ_h, with left endpoints starting at 0, $\delta\ell_h$, $2\delta\ell_h$, $3\delta\ell_h$, ..., in total, about $N/(\delta\ell_h)$ intervals.

How much time will it take to compute the confidence of each such interval? Let us just compute the sum of the lengths of the intervals, and multiply at the end by $g(N)/N$. For each of the $\log_{1+\delta} N$ values h, there are $N/(\delta\ell_h)$ intervals, each of length ℓ_h. Hence their sum of lengths is N/δ. It follows that the sum of their lengths is the number of h's, i.e., $\log_{1+\delta} N$, times N/δ. Since $\log_{1+\delta} N$ is approximately $(\lg N)/\delta$ for small δ, the product is $(N \log N)/\delta^2$.

However, we can do better. To date we have used only the containment property; now we use the prefix property. We modify the intervals' design so that many will have the same left endpoint. Break the intervals into groups according to their lengths: those with lengths in $[1, 2)$, those with lengths in $[2, 4)$, those with lengths in $[4, 8)$, etc. There are obviously $\lg N$ groups. Within a group, our intervals have length ℓ_h for varying h's; their left endpoints are multiples of $\delta\ell_h$. We now change their left endpoints as follows. For intervals with lengths in $[A, 2A)$, now make the left endpoints multiples of $\delta A \leq \delta\ell_h$ (rather than $\delta\ell_h$), shrinking the gap between consecutive left endpoints and enlarging the number of intervals by less than a factor of 2. However, note the following important fact: all the intervals with lengths in $[A, 2A)$ start at $0, \delta A, 2\delta A, 3\delta A$,.... By the prefix property, it suffices to include in the running time only the length of the longest interval with a given starting point. Hence we can process all the intervals with lengths in $[A, 2A)$ in time $g(N)/N$ multiplied by $O(N/(\delta A))(2A)$, which is $g(N)/N$ times $O(N/\delta)$. Since there are only $\lg N$ such groups (and not $\log_{1+\delta} N$, as before), the total time to process all intervals will be $g(N)/N$ times $O((N \lg N)/\delta)$. Hence, for LIS computation, for example, for which $g(N)/N$ is $O(\log N)$, the overall time will be $O((N \lg^2 N)/\delta)$.

CLAIM 13 *Let $\mathcal{I}$ be the set of intervals in an optimal solution, each having confidence at least $\hat{c}$, and $\mathcal{J}$ be the set of intervals considered by our algorithm. For each $I \in \mathcal{I}$, there exists a $J \in \mathcal{J}$ with confidence $\geq \left(\frac{1-\delta}{1+\delta}\right)\hat{c}$ containing I.*

PROOF. How small a δ must we use such that for any interval $I = [a, b] \subseteq [0, N]$ of length at least 1, one of our intervals contains I and has length at most $1+\epsilon$ times as large? Choose h smallest such that $\ell_h - \delta\ell_h \geq b - a$, i.e., $\ell_h \geq (b-a)/(1-\delta)$. Then one of our intervals starts at a or no more than $\delta\ell_h$ to the left of a, and ends at or to the right of b. That interval contains I, clearly. By minimality of h, $\ell_{h-1} < (b-a)/(1-\delta)$, and therefore the length $(1+\delta)^h$ of our interval is at most $(1+\delta)/(1-\delta)$ times I's length, proving Claim 13. □

Claim 13 implies that it suffices to choose δ small enough that $(1+\delta)/(1-\delta) \leq 1+\epsilon$, i.e., $\delta \leq \epsilon/(2+\epsilon)$.

(For brevity, we have omitted some implementation details on converting the real intervals into sets of contiguous integers.)

3.2 Tableau Assembly

Given a set of intervals in $[0, N]$ satisfying the confidence threshold, each with integral endpoints and no two with the same left endpoint, we can assemble a tableau T_r with support at least $\hat{s}$ by selecting enough intervals to cover the desired number of points; in particular, we wish to choose the minimal number of intervals needed. Each selected (position) interval $[i..j]$ then determines the tableau pattern $[t_{\pi(i)}[X], t_{\pi(j)}[X]]$, i.e., the position interval mapped back to a range of X-values. We first show that, unlike the more general PARTIAL SET COVER problem, our problem is in $\mathcal{P}$, by exploiting the fact that we have intervals rather than arbitrary sets. We give a $O(N^2)$-time dynamic programming algorithm to find a minimal (partial) cover. The algorithm takes as input a set $\mathcal{J}$ of intervals of the form $[i..j] = \{i, i+1, ..., j\}$, for some $1 \leq i, j \leq N$, and assumes they are sorted on their left endpoints. Via dynamic programming, the algorithm computes, for each $0 \leq k, \ell \leq N$, the value $T(k, \ell)$ which equals the minimum number of the given intervals necessary to cover at least k points among $\{1, 2, ..., \ell\}$ (or ∞ if it is not possible to do so); the final answer is $T(\lceil \hat{s}N \rceil, N)$. $T(0, 0) = 0$ and $T(k, 0) = \infty$ for all $k > 0$. After $T(k, \ell')$ has been computed for all $\ell' < \ell$ and all $k = 0, 1, 2, ..., N$, the algorithm computes $T(k, \ell)$ for all $k = 0, 1, 2, ..., N$, using Lemma 14.

[1] The perceptive reader may have noted a discrepancy between the *size* $j - i + 1$ of a position interval $[i, j]$ and the *length* $j - i$ of the real interval $[i, j]$. Since it is more natural to work with real intervals, we just "equate" the set $\{i, i+1, ..., j\}$ with the real interval $[i - 0.5, j + 0.5]$. The length of any union of such real intervals equals the size of the corresponding union of sets of integers. This allows us to continue working with real intervals with impunity.

LEMMA 14 *If there is no input interval containing ℓ, then $T(k,\ell) = T(k,\ell-1)$. Otherwise, among all intervals containing ℓ, choose the one whose left endpoint is smallest; denote its left endpoint by $\ell - z + 1$. Then*

$$T(k,\ell) = \min\{T(k,\ell-1), 1 + T(k-z,\ell-z)\}.$$

PROOF. As the first statement is obvious, we move on to the second. The optimal way to cover at least k of the points $1, 2, ..., \ell$ either covers the point ℓ or it does not. If it does not, its cost is $T(k,\ell-1)$. If it does, it contains some interval which contains ℓ. Without loss of generality it contains, among those intervals containing ℓ, the one whose left endpoint is as small as possible. Suppose that that interval has left endpoint $\ell - z + 1$ and therefore covers the z points $\ell - z + 1, \ell - z + 2, ..., \ell$. Then $T(k,\ell) = T(k-z,\ell-z) + 1$. □

Lemma 14 suggests an easy $O(N^2)$-time algorithm for computing all the $T(k,\ell)$ values. Since the quadratic complexity of the dynamic programming algorithm makes it infeasible for large data sets, we consider an approximation to find a nearly minimal size using a greedy algorithm for PARTIAL SET COVER (see [18]). We show that, for the special case in which the sets are intervals, the algorithm can be implemented in linear time and provides a constant performance ratio.

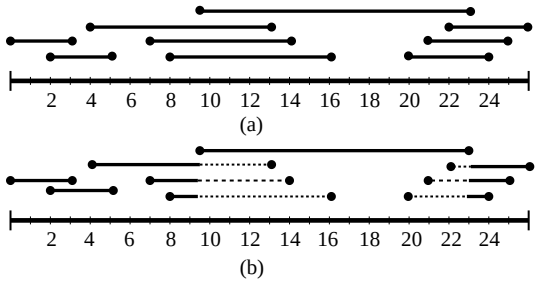

Figure 3: Adjusting marginal cardinalities.

CLAIM 15 *The greedy partial set cover algorithm can be implemented to run in time $O(N)$.*

PROOF. A set of intervals is given sorted on left (and also right) endpoints by the candidate generation phase. We separately maintain these intervals ordered by set cardinality in an array $1..N$ of linked lists, where the array index corresponds to cardinality. At each step, we iterate down (initially from N) to the largest index containing a non-empty linked list, to find an interval with the largest "marginal" cardinality (which only counts points that have not already been covered by an interval that has already been added to the tableau), and adjust the marginal cardinalities of any overlapping intervals. Consider the intervals shown in Figure 3 (a) and suppose that the longest one has just been added to the tableau. As seen in Figure 3 (b), six intervals need to have their marginal cardinalities updated. Further, of these six intervals, which are now shorter, four are now contained in other ones and may be deleted. In general, each iteration of the algorithm deletes all but one interval intersecting the left endpoint of the currently longest interval; likewise for the right endpoint. Since there are at most N iterations and we adjust at most two intervals per iteration, the time spent adjusting the nondeleted intervals is $N * O(1) = O(N)$. The total time spent deleting intervals, over the entire execution of the algorithm, is $O(N)$, since there are at most N intervals.

CLAIM 16 *The greedy algorithm gives a constant performance ratio. (See the Appendix for a proof.)*

An important property of our framework is that the size of a generated tableau can be no larger than the tableau generated when there is no cheating on confidence in the candidate interval phase, given the same confidence threshold. This is easy to see because cheating on confidence can only yield intervals subsuming optimal intervals, and with better choices available an optimal (partial) set cover will be at most as large.

3.3 Examples of Confidence Metrics

First, we show that our tableau generation framework is compatible with our definition of confidence (Definition 2). In the special case of "simple" CSDs (recall Section 2.2.1), we need to compute the length of a LIS in a given interval in order to compute its confidence. Many implementations of longest increasing subsequence incrementally maintain LIS on increasing prefixes in $O(N \log N)$ time; hence, LIS satisfies the prefix property. As for the containment property, clearly if one interval is contained in another, then any subsequence of the smaller interval must be contained in the larger. Therefore, for simple CSDs, our framework is able to find candidates in $O((N \log^2 N)/\delta)$ time.

While there is work on simultaneously computing LIS's of multiple (overlapping) windows of a sequence [2, 6], none of this work breaks the quadratic complexity barrier. Recent work on computing the approximate size of a longest increasing subsequence on streams saves space but not time [14]. Hence, we are not aware of a faster way to compute LIS that can help in our context.

The dynamic program from Section 2.2 provides values at every prefix en route to computing the confidence of the entire interval, thus satisfying the prefix property. The containment property is also satisfied because the same valid gap sequence converted from an interval would also be available to any interval containing it; it would require no more deletions than the difference in the lengths to transform the larger interval into the same valid gap sequence. So for general CSDs, our framework is able to find candidates in $O(\frac{G_2}{G_2-G_1}(N \log^2 N)/\delta)$ time. We know of no existing work that could help improve the time complexity for general CSDs.

If one prefers to define confidence differently, such as based on the average number of inversions for SDs of the form $X \rightarrow_{[0,\infty)} Y$, or based on the fraction of gaps within $[G_1, G_2]$ for SDs of the form $X \rightarrow_{[G_1,G_2]} Y$ with $G_2 < \infty$, then our framework also applies. We leave it as an exercise to verify that the former can be computed in time $O(N \log^2 N/\delta)$ and the latter in $O(N \log N/\delta)$ using our framework.

4. EXPERIMENTS

In this section, we present an experimental evaluation of our proposed tableau discovery framework for conditional sequential dependencies, which comprises candidate interval generation (CANDGEN) and tableau assembly (TABASSMB). First, to justify the motivation and utility of CSDs, we present sample tableaux which unveil interesting data semantics and potential data quality problems. Second, for both CANDGEN and TABASSMB, we investigate the trade-off between tableau quality and performance of resorting to approximation. Finally, we demonstrate the efficiency and scalability of the proposed tableau generation framework.

4.1 Data Sources and Platform

Experiments were performed on a 2.7 GHz dual-core Pentium PC with 4 GB of RAM. The performance numbers presented are based on real time as reported by the Unix `time` command. Ex-

periments were run 5 times and the average time was reported. All algorithms were implemented in C++.

We used the following four data sources for our experiments. Table 1 displays a summary of data characteristics.

- DOWJONES consists of daily closing figures of the *Dow Jones Industrial Average*[2], and has the schema (DATE, AVGCLOSING). The closing figures have been smoothed using a 2-week moving window average.
- WEATHERDATES consists of the days on which daily temperatures were recorded at Gabreski Airport[3] in Long Island, NY, from 1943.07.18 - 2008.10.01 by *Global Summary of the Day*[4].
- NETWORKFEEDS consists of data feeds of probed measurements from an ISP and the associated timestamps when they were received.
- TRAFFICPOLLS contains the timestamps of traffic volume measurements in an ISP that were configured to be taken every 5 minutes.

DATASET	#TUPLES	DEPENDENCY
DOWJONES	27399	DATE $\rightarrow_{(0,\infty)}$ AVGCLOSING
NETWORKFEEDS	916961	STARTTIME $\rightarrow_{(0,\infty)}$ ENDTIME
WEATHERDATES	15716	ARRIVALORDER$\rightarrow_{[0,1]}$DATE
TRAFFICPOLLS	91522	ARRIVALORDER$\rightarrow_{[270,330]}$TIME

Table 1: Summary of Data Sources

In the experiments that follow, we use the confidence threshold $\hat{c} = 0.995$, support threshold $\hat{s} = 0.5$ (note that the tableau assembly algorithm may terminate before reaching the required support if it runs out of candidate patterns), and approximation tolerance parameter $\delta = 0.05$, unless mentioned otherwise.

4.2 Sample Tableaux

We first show that CSDs with different gap values can capture interesting semantics. We also show that our approximate framework discovers tableaux that are close to optimal.

Table 2 compares tableaux generated by exhaustive candidate generation (EXACTINTVL) and our approximate candidate generation (APPRXINTVL), for various gaps with greedy TABASSMB on the WEATHERDATES dataset. The support of each pattern is also shown, indicating the number of data values contained in the corresponding interval. Gap ranges of $[0, 1]$ (at least one temperature reading per day) and $[0, 2]$ (one reading every two days) result in tableaux with two rows, indicating that there was at least one major break in the data recording. Note that the exact and approximate tableaux "latch onto" different endpoints. This was due to δ being set to 0.05, which meant that a confidence threshold of 0.995 was used for the exact tableau whereas effectively $0.995(1 - 0.05)/(1 + 0.05) = 0.9$ was used for the approximate one. When we used $\delta = 0.01$ for the gap range $[0, 2]$, the approximate tableau was the same as the exact one.

Next, we identify time ranges over different scales over which no temperature data was recorded. A gap range $[2, 10]$ was used to find periods when the recording was discontinued for about ten days at a time, possibly due to malfunctioning equipment. A comparison of the tableau row start and end dates, as well as their associated supports, reveal that the exact and approximate tableaux were quite similar, and both indicate periods when no data was recorded. A gap range of $[6, 10]$ helps identify a time-frame from $1991.01.01$ to $1991.01.08$ which has 6 days of missing data. (since the support is 2, only the beginning point and the endpoint are present in the data). Similarly, $[10, 20]$ returned 6 periods of moderate data loss—ten to 20 days at a time. In order to capture regions of long gaps, a gap range of $[20, \infty)$ was used. The first two patterns identify the two time periods of most significant loss: 1945 to 1951 and 1969 to 1980, when, according to the Wikipedia page for this airport given in Footnote 3, it was closed to the public.

EXACTINTVL Tableau	Support	APPRXINTVL Tableau	Support
Gap:[0,1]			
Tableau size: 2		Tableau size: 2	
1945.11.06 - 1969.12.15	6819	1944.02.07 - 1981.01.23	7536
1980.12.09 - 1990.12.10	3636	1981.02.05 - 2006.02.04	6999
Gap:[0,2]			
Tableau size: 2		Tableau size: 2	
1945.11.01 - 1969.12.15	6824	1944.02.07 - 1981.01.29	7542
1980.10.22 - 1991.01.23	3681	1981.02.05 - 2006.07.31	7176
Gap:[0,5]			
Tableau size: 2		Tableau size: 1	
1945.10.29 - 1969.12.15	6827	1943.07.18 - 2008.05.25	15588
1980.11.23 - 1995.05.10	5115		
Gap:[2,10]			
Tableau size: 20		Tableau size: 20	
1995.06.23 - 1995.06.28	3	1995.06.23 - 1995.06.28	3
1983.02.21 - 1983.02.23	2	1983.02.21 - 1983.02.23	2
1985.09.27 - 1985.10.01	2	1985.09.27 - 1985.10.01	2
1988.04.05 - 1988.04.08	2	1988.04.05 - 1988.04.08	2
Gap:[6,10]			
Tableau size: 1		Tableau size: 1	
1991.01.01 - 1991.01.08	2	1991.01.01 - 1991.01.08	2
Gap:[10,20]			
Tableau size: 6		Tableau size: 6	
1951.06.01 - 1951.06.12	2	1951.06.01 - 1951.06.12	2
1980.11.23 - 1980.12.09	2	1980.11.23 - 1980.12.09	2
1990.12.20 - 1991.01.01	2	1990.12.20 - 1991.01.01	2
1991.01.11 - 1991.01.23	2	1991.01.11 - 1991.01.23	2
1991.02.18 - 1991.03.01	2	1991.02.18 - 1991.03.01	2
1994.07.15 - 1994.07.27	2	1994.07.15 - 1994.07.27	2
Gap:$[20, \infty)$			
Tableau size: 9		Tableau size: 9	
1945.11.29 - 1951.04.30	2	1945.11.29 - 1951.04.30	2
1969.12.15 - 1980.10.22	2	1969.12.15 - 1980.10.22	2
1991.10.02 - 1991.10.30	2	1991.10.02 - 1991.10.30	2
1993.10.19 - 1993.11.16	2	1993.10.19 - 1993.11.16	2
1994.02.03 - 1994.03.01	2	1994.02.03 - 1994.03.01	2
1995.05.10 - 1995.06.22	2	1995.05.10 - 1995.06.22	2

Table 2: Tableau sizes for various gap values on WEATHERDATES

Table 3 presents the sample tableaux for TRAFFICPOLLS. The expected time gap between two successive polls is 5 minutes, or 300 seconds. Due to several factors from traffic delays to clock synchronization, this exact periodicity will hardly ever be met. Therefore, we allow for ± 30 seconds and use a gap of 270 seconds to 330 seconds. The gap range $[270, 330]$ is satisfied by much of the data and gives a tableau size of two. Next, a gap range of $[0, 150]$ was used to identify regions of extraneous polls. There are several instances of very short time differences between polls, but these tend to occur only briefly (one poll). A gap range of $[350, \infty)$ was then used to identify regions with heavily delayed or missing data, which, when correlated with other information collected by the ISP, helped solve problems with the data collection mechanism.

Table 4 presents sample tableaux for different gap ranges on the DOWJONES data set. Patterns for $(0, \infty)$ show time ranges over which Dow Jones stock market exhibited an increasing trend with

[2]www.economagic.com/sp.htm

[3]http://en.wikipedia.org/wiki/Francis_S._Gabreski_Airport

[4]www.ncdc.noaa.gov/cgi-bin/res40.pl?page=climvisgsod.html

APPRXINTVL Tableau	Support
Gap:$[270, 330]$ Tableau size: 2	
2008-10-09,05:17:06 - 2009-03-06,19:10:17	39925
2008-04-22,23:15:38 - 2008-05-25,21:33:20	8683
Gap:$[0, 150]$ Tableau size: 751	
2008-03-17,21:07:02 - 2008-03-17,21:08:56	2
2008-04-14,16:00:23 - 2008-04-14,16:00:23	2
2008-05-26,05:05:31 - 2008-05-26,05:05:31	2
2008-05-26,06:17:43 - 2008-05-26,06:17:47	2
Gap:$[350, \infty)$ Tableau size: 4001	
2008-09-14,13:18:38 - 2008-09-14,14:59:24	14
2008-09-17,07:33:51 - 2008-09-17,09:06:06	11
2008-09-17,01:22:46 - 2008-09-17,02:11:02	7
2008-04-13,03:48:21 - 2008-04-13,05:32:38	6

Table 3: Tableaux for TRAFFICPOLLS

very high confidence of 0.995.

The first few patterns for gap $[0, 5]$ are similar to those of $(0, \infty)$. This implies that successive increases in stock market prices, particularly over long periods of time, are usually by small amounts which mostly lie within the small range of $[0, 5]$.

Gaps $[50, 100]$ and $[100, \infty)$ capture regions where the stock market average closing price increased rapidly. The resulting tableau suggests that sharp increases in stock prices were mostly observed during the late nineties and early years of the 21^{st} century, probably due to the "dotcom boom" and "housing bubble".

APPRXINTVL Tableau	Support	APPRXINTVL Tableau	Support
Gap:$[0, \infty)$ Tableau size: 246		Gap:[0,5] Tableau size: 286	
1949.06.07 - 1950.06.22	261	1949.06.07 - 1950.06.22	261
1904.05.17 - 1905.04.26	237	1904.05.17 - 1905.04.26	237
1921.08.15 - 1922.06.13	206	1921.08.15 - 1922.06.13	206
1953.09.18 - 1954.06.08	179	1953.09.18 - 1954.06.08	179
1942.07.28 - 1943.04.12	176	1942.07.28 - 1943.04.12	176
1925.03.24 - 1925.11.16	166	1925.03.24 - 1925.11.16	166
1915.05.19 - 1916.01.07	162	1915.05.19 - 1916.01.07	162
1898.09.30 - 1899.05.05	149	1898.09.30 - 1899.05.05	149
1958.04.11 - 1958.10.24	138	1935.03.14 - 1935.09.24	135
1935.03.14 - 1935.09.24	135	1945.07.26 - 1946.02.13	134
Gap:[50,100] Tableau size: 45		Gap:$[100, \infty)$ Tableau size: 4	
2000.10.27 - 2000.11.08	9	2000.03.20 - 2000.03.28	7
1998.10.14 - 1998.10.23	8	2001.04.17 - 2001.04.18	2
1999.03.10 - 1999.03.18	7	2002.10.18 - 2002.10.21	2
2001.04.24 - 2001.05.01	6	2002.10.22 - 2002.10.23	2
2001.11.08 - 2001.11.19	6		
2002.03.01 - 2002.03.08	6		
2003.03.19 - 2003.03.26	6		
1999.04.13 - 1999.04.16	4		
1999.04.20 - 1999.04.23	4		
1999.07.08 - 1999.07.13	4		

Table 4: Tableaux for DOWJONES

4.3 Quality

In this section, we reinforce our claims of good tableau quality by comparing EXACTINTVL and APPRXINTVL tableaux over a wide variety of $\hat{c}$, $\hat{s}$ and δ values. Since it is impractical to present actual tableaux for all the aforementioned cases, we use tableau size as a substitute for tableau quality and compare tableau sizes instead. Figure 4 demonstrates the quality results of our approximate algorithms for the DOWJONES data.

For the gap range $[0, \infty)$, Figure 4(a) compares the tableau sizes obtained from candidate intervals generated by EXACTINTVL and APPRXINTVL (using different δ's), as a function of confidence threshold $\hat{c}$, using support $\hat{s} = 0.5$. (Exact tableau assembly was used on the candidates from both methods.) The tableau sizes are quite similar for low values of $\hat{c}$, and for high values of $\hat{c}$ with low values of δ. For example, at $\hat{c} = 0.8$ with $\delta = 0.01$, there was only a difference in size of 12. For the gap range $[0, 5]$, Figure 4(d) shows a greater sensitivity to δ, as there was a much more pronounced difference in tableau sizes at $\delta = 0.05$, but again for $\delta = 0.01$ the difference was relatively small.

In the previous experiments, although a desired confidence threshold of $\hat{c}$ was supplied, the algorithm relaxes this value to as low as $\left(\frac{1-\delta}{1+\delta}\right)\hat{c}$ to guarantee that all optimal candidate intervals are covered by some interval reported by the approximation algorithm. Hence, the tableau size is never larger, and may be smaller, than that of an optimal solution. However, if one does not wish to allow such "cheating on confidence", then an alternative is as follows. Instead, we can "inflate" the desired confidence from $\hat{c}$ to $\min\left(1, \left(\frac{1+\delta}{1-\delta}\right)\hat{c}\right)$ so that the relaxed confidence is now no less than $\hat{c}$ (but no greater than one), and thus not "cheat". Of course, this may now result in larger tableaux than the optimal. As usual, the effect of this will depend on δ: when δ is small, $\hat{c}$ will only be inflated by a small amount and thus the tableau sizes will be closer to optimal. The trade-off is that the algorithm takes longer with smaller values of δ but, as we shall show in the next subsection when we investigate performance, even with relatively small values of δ there is a significant improvement over running the exact algorithm.

Figures 4(b) and 4(e) compare the results of

- $OPTIMAL$ (EXACTINTVL with unmodified $\hat{c}$),
- $apprx$ (APPRXINTVL with inflated $\hat{c}$)
- $optimal$ (EXACTINTVL with inflated $\hat{c}$)

with various δ-values. Note that the tableau size of $OPTIMAL$ lower-bounds that of $apprx$, which lower-bounds that of $optimal$. Figure 4(b) assumes gaps of $[0, \infty)$, whereas Figure 4(e) uses gaps of $[0, 5]$. In all cases, the tableau sizes for APPRXINTVL (with inflated $\hat{c}$) are lower-bounded by $OPTIMAL$ (with unmodified $\hat{c}$). This implies that, in order to obtain an "exact" tableau (using EXACTINTVL) for a given $\hat{c}$, one might as well assume an inflated confidence of $\hat{c}\frac{1+\delta}{1-\delta}$ (not exceeding 1) and obtain the same results using the much faster APPRXINTVL CANDGEN. Similar behavior was observed for other $\hat{s}$ values. For higher $\hat{s}$, the absolute value of the tableau sizes increase, as expected.

Figures 4(c) and 4(f) show that the greedy set cover algorithm (GREEDYASMBL) gives similar sized tableaux compared to the optimum that dynamic programming (EXACTASMBL) obtains—the curves are almost indistinguishable—at a variety of support thresholds with $N = 10000$. Figure 4(c) assumes gaps of $[0, \infty)$ and $\hat{c} = 0.63$, whereas Figure 4(f) uses gaps of $[0, 5]$ and $\hat{c} = 0.7$. Both the exact and approximate TABASSMB algorithms ran on the same set of input candidate intervals, which were generated by APPRXINTVL. Similar figures were obtained for other values of $\hat{c}$ and other data sets.

4.4 Scalability

The previous two subsections highlight the fact that tableaux generated by APPRXINTVL are close to that of EXACTINTVL. In this subsection, we show that APPRXINTVL can generate accurate tableaux while still being faster than EXACTINTVL by orders of magnitude. For the sake of efficiency on large inputs, all tableau

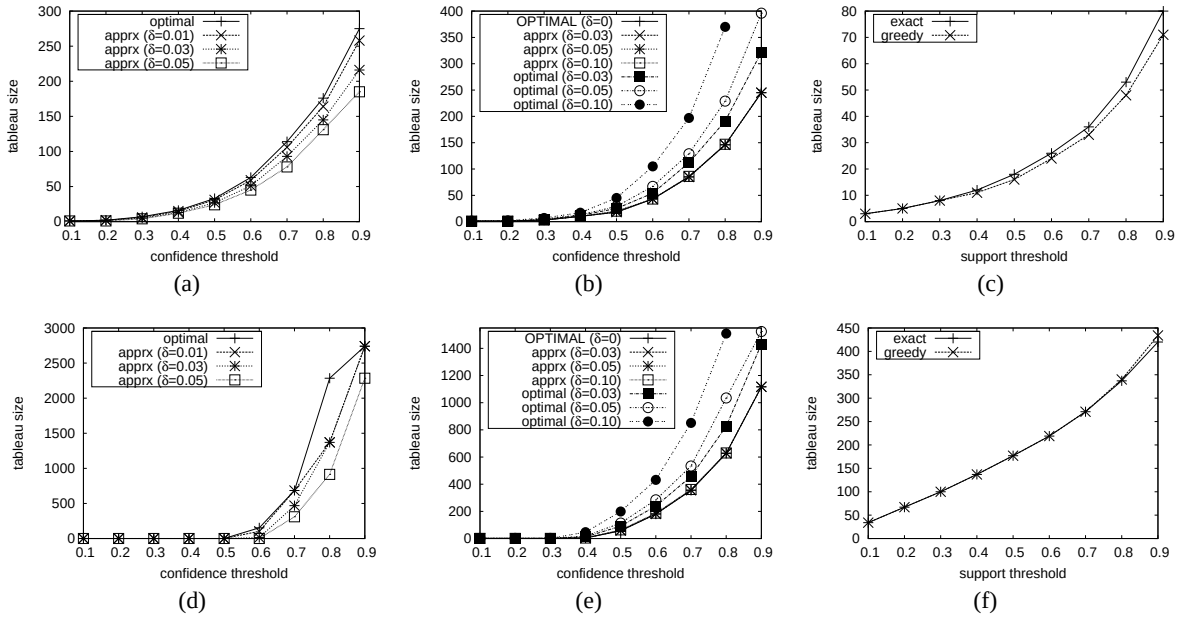

Figure 4: Tableau sizes obtained for DOWJONES using: (a) EXACTINTVL vs APPRXINTVL at different $\hat{c}$ for $[0, \infty)$ (b) at different inflated $\hat{c}$ for $[0, \infty)$ (c) EXACTASMBL vs GREEDYASMBL at different $\hat{s}$ for $[0, \infty)$ ($\hat{c} = 0.63$) (d) EXACTINTVL vs APPRXINTVL at different $\hat{c}$ for $[0, 5]$ (e) at different inflated $\hat{c}$ for [0,5] (f) EXACTASMBL vs GREEDYASMBL at different $\hat{s}$ for $[0, 5]$ ($\hat{c} = 0.7$)

generation methods in this section use GREEDYASMBL for assembly; results are reported as combined running time of CANDGEN and TABASSMB phases.

Figure 5 compares the performance of APPRXINTVL (using various δ-values) with EXACTINTVL for different data set sizes. For the gap range $[0, \infty)$, Figure 5(a) and Figure 5(b) present results using DOWJONES and NETWORKFEEDS data, respectively. APPRXINTVL scales more gracefully, especially in Figure 5(b) where we had to halt the exhaustive algorithm because it ran too long. Figure 5(e) and Figure 5(f) show results using WEATHERDATES with gaps in $[4, 6]$ and TRAFFICPOLLS with gaps in $[270, 330]$, respectively. As before, the APPRXINTVL algorithm results in substantial improvement over EXACTINTVL, particularly for large number of inputs as can be seen in Figure 5(f). While the performance is noticeably better with larger δ-values, in all cases the performance of APPRXINTVL is much faster (orders of magnitude) than that of EXACTINTVL, even with very low values of δ (say, 0.01).

Figure 5(c) and 5(d) separate out the performance comparison of CANDGEN and TABASSMB phases, using DOWJONES data. Figure 5(c) compares the running times of APPRXINTVL with EXACTINTVL and 5(d) compares GREEDYASMBL with EXACTASMBL. In Figure 5(d), the curve for GREEDYASMBL is indistinguishable from the x-axis.

5. RELATED WORK

Since database relations are abstracted as (multi)sets, classical integrity constraints do not directly apply to ordered data. One of the few proposed constraints to do so is the order dependency (OD) proposed in [11] and the related notion of *sort sets* [12], approximate versions of which were studied in [8]. This paper is the first to propose Sequential Dependencies, which generalize Order Dependencies and express a wider range of practical order semantics, and to study their approximate and conditional variants.

Recently, traditional integrity constraints have been adapted to apply *conditionally* on the data to be able to capture the semantics of, and errors commonly found in, real data. Conditional Functional Dependencies (CFDs) were proposed in [4]. Conditional Inclusion Dependencies were proposed in [5]. In a similar vein, we have proposed Conditional Sequential Dependencies to express the sequential semantics commonly found in ordered data and to detect potential data quality problems.

The problem of *discovering* dependencies in data was first studied in [17, 19] where the goal was to find antecedent and consequent attributes from among different subsets of attributes in the schema satisfying an FD. The problem of CFD tableau discovery, given an embedded FD, was introduced in [13]. The problem of discovering CFDs (FDs as well as tableaux) was studied in [7, 9] but used different criteria (e.g., no global support threshold). Our tableau discovery framework is most closely related to that proposed in [13]. However, [13] deals with CFDs, which do not have any sequential semantics. Moreover, the *range tableaux* proposed in [13] specify that the FD independently holds on each subset of tuples that agree on the antecedent attributes such that the value of the ordered attribute is within the range. In this paper, a tableau pattern denotes that the underlying SD holds in the entire interval. Thus, while our definitions of confidence and tableau "goodness" are analogous to those of [13], it is not possible to express SDs using FDs (with or without range tableaux), and our pattern generation and tableau discovery algorithms are novel and non-trivial.

Constraints pertaining to ordering have also been considered in the temporal database literature (e.g., [15]) but focus on satisfiability and implication problems, which are orthogonal to our work. Closest to our work from this literature is [24] which, given a dependency $X \rightarrow Y$, discovers operators in $\{<, =, >\}$ (approximately) satisfying the dependency over the entire data set (using a different confidence metric); conditioning and tableaux were not considered. Other sequential constraints have been studied for data streams such as PACs [20], which monitor properties such as poll time differences are approximately 5 minutes, with high probabil-

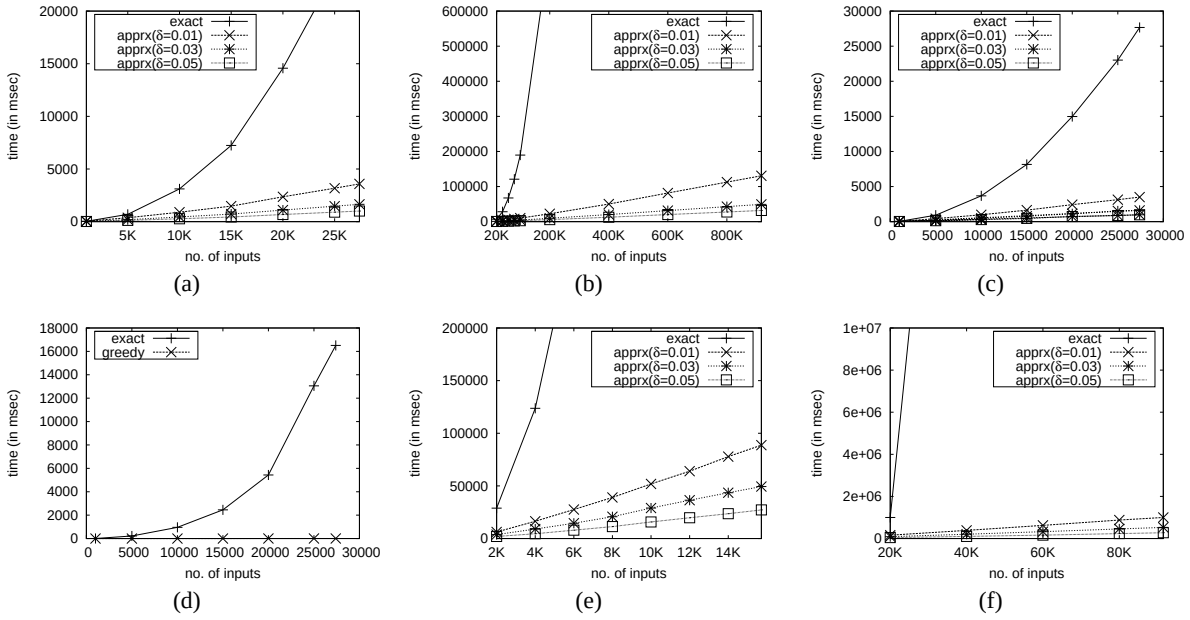

Figure 5: Scalability: (a) DOWJONES ($\hat{c} = 0.63, \hat{s} = 0.5$) (b) NETWORKFEEDS ($\hat{c} = 0.99, \hat{s} = 0.5$) (c) EXACTINTVL and APPRXINTVL CANDGEN for DOWJONES (d) EXACTASMBL vs GREEDYASMBL TABASSMB for DOWJONES (e) WEATHERDATES ($[4, 6]$ gap range, $\hat{c} = 0.9, \hat{s} = 0.5$) (f) TRAFFICPOLLS ($[270, 330]$ gap range, $\hat{c} = 0.9, \hat{s} = 0.5$)

ity; and k-constraints [3], which monitor properties such as a SYN packet must be followed by an ACK within k packets in a TCP stream. Neither considers conditioning or discovery.

Finally, there is some marginally related work in the area of data mining including sequential association rules [1], which find frequent subsequences contained in a sequence; frequent episodes [21], which find partially ordered event subsequences that occur in a small time window; and correlations between attributes [22, 25]. The patterns they find are different from SDs and none of this work attempts to capture the semantics of a large portion of the data via a global support threshold.

6. CONCLUSIONS

Detecting data quality problems and understanding data semantics are challenging in practice. Fortunately, there are some order semantics implicit in sequential data that can be leveraged. Here we have initiated the study of *Sequential Dependencies*, which are integrity constraints that define dependencies between two sets of attributes in terms of their co-orderings. We proposed a framework for efficiently discovering tableaux for Conditional (Approximate) Sequential Dependencies that satisfy specified support and confidence constraints and are parsimonious; our framework runs in sub-quadratic time when the function for computing confidence obeys certain properties. We have illustrated the efficiency and utility of our framework on a variety of real data sets.

Since the mechanisms generating ordered data often suggest the underlying order semantics, we have assumed that the underlying sequential dependency is known. An interesting direction for future work is to discover reasonable sequential dependencies from a relation instance or a sample. We are also interested in studying the properties of sets of SDs and CSDs, including axiomatization, inference and satisfiability. Finally, an aspect of our framework for tableau discovery that remains to be studied is how to efficiently maintain tableaux in the presence of updates to the base relation.

7. REFERENCES

[1] R. Agrawal and R. Srikant. Mining sequential patterns. *ICDE* 1995, 3–14.

[2] M. Albert et al. Longest increasing subsequences in sliding windows. *Theor. Comput. Sci.*, 321(2-3):405–414, 2004.

[3] S. Babu, U. Srivastava, and J. Widom. Exploiting k-constraints to reduce memory overhead in continuous queries over data streams. *ACM Trans. Database Syst.*, 29(3):545–580, 2004.

[4] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. *ICDE* 2007, 746–755.

[5] L. Bravo, W. Fan, and S. Ma. Extending dependencies with conditions. *VLDB* 2007, 243–254.

[6] E. Chen, H. Yuan, and L. Yang. Longest increasing subsequences in windows based on canonical antichain partition. *Theor. Comput. Sci.* 378(3):223–236, 2007.

[7] F. Chiang and R. Miller. Discovering data quality rules. *PVLDB*, 1(1):1166–1177, 2008.

[8] J. Dong and R. Hull. Applying approximate order dependency to reduce indexing space. *SIGMOD* 1982, 119–127.

[9] W. Fan, F. Geerts, L.V.S. Lakshmanan, and M. Xiong. Discovering conditional functional dependencies. *ICDE* 2009, to appear.

[10] M. Fredman. On computing the length of longest increasing subsequences. *Discrete Math.*, 11:29–35, 1975.

[11] S. Ginsburg and R. Hull. Order dependency in the relational model. *Theor. Comput. Sci.*, 26:149–195, 1983.

[12] S. Ginsburg and R. Hull. Sort sets in the relational model. *PODS* 1983, 332-339.

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

[14] P. Gopalan, T. Jayram, R. Krauthgamer, and R. Kumar. Estimating the sortedness of a data stream. *SODA* 2007, 318–327.

[15] S. Guo, W. Sun, and M. Weiss. Solving satisfiability and implication problems in database systems. *ACM Trans. Database Syst.*, 21(2):270–293, 1996.

[16] A. Gupta and F. Zane. Counting inversions in lists. *SODA* 2003, pages 253–254.

[17] Y. Huhtala, J. Karkkainen, P. Porkka, and H. Toivonen. Tane: An

efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42(2):100–111, 1999.

[18] M. Kearns. *The Computational complexity of machine learning*. MIT Press, 1989.

[19] J. Kivinen and H. Mannila. Approximate inference of functional dependencies from relations. *Theor. Comput. Sci.*, 149(1):129–149, 1995.

[20] F. Korn, S. Muthukrishnan, and Y. Zhu. Checks and balances: monitoring data quality problems in network traffic databases. *VLDB* 2003, 536–547.

[21] H. Mannila, H. Toivonen, and A. Verkamo. Discovery of frequent episodes in event sequences. *Data Min. Knowl. Discov.*, 1(3):259–289, 1997.

[22] S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming pattern discovery in multiple time-series. *VLDB* 2005, 697–708.

[23] M. Stonebraker and U. Cetintemel. "one size fits all": An idea whose time has come and gone. *ICDE* 2005, 2–11.

[24] J. Wijsen. Temporal dependencies with order constraints. Technical report, TimeCenter TR-47, 1999.

[25] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. *VLDB* 2002, 358–369.

APPENDIX

We are studying the following problem. Given a set of at most n closed real intervals, no one a subset of any other, and a real L, run the greedy PARTIAL SET COVER algorithm until the coverage is at least L. (*Coverage* is the length of the union of all chosen intervals.)

Consider a set of the fewest input intervals which has coverage at least L. Call the intervals in the solution *opt intervals*; call those obtained by the greedy algorithm *greedy intervals*. Let t be the number of opt intervals.

We need to study a game involving an adversary. Given are t nonnegative reals $g_1^0, g_2^0, ..., g_t^0$. Define $G_0 := \sum_{i=1}^t g_i^0$. Set $g_i := g_i^0$ for all i. In addition, there is a real C which is initially 0. Repeatedly the adversary makes either a type-1 move or a type-2 move.

In a *type-1 move*, the adversary specifies a δ_i for each i, $1 \le i \le t$, with $0 \le \delta_i \le g_i$. Each g_i is decreased by δ_i and C is increased by at least $\sum \delta_i$.

In a *type-2 move*, no g_i changes, but C increases by some amount $\delta \ge \max_{i=1}^t g_i$. Let δ be the *cost* of this type-2 move.

LEMMA 17 *Suppose that all $g_i^0 = A$, $1 \le i \le t$, for some given real $A \ge 0$. Then however the game is played, by the time the adversary has made at least T moves of type 2, $C \ge G_0 = tA$, if $T \ge t$.*

PROOF. Focus on two different type-2 moves with no type-2 moves between them. In the block of moves strictly between the two type-2 moves, the adversary makes some number, possibly zero, of type-1 moves. During that block of type-1 moves, if $\sum_i g_i$ decreases from a to b, then C increases by at least $a - b$.

After exactly l type-2 moves to date have been performed, let Z_l denote the sum of the costs of the l type-2 moves to date. Let g_i^l be the value of g_i at the time when the lth type-2 move is performed. By induction, it is easy to prove that after l type-2 moves have been performed,

$$C \ge \sum_{i=1}^{t}(g_i^0 - g_i^l) + Z_l \ge \sum_{i=1}^{t}(g_i^0 - \max_{j=1}^{t} g_j^l) + Z_l = t(A - \max_{i=1}^{t} g_i^l) + Z_l. \quad (1)$$

Let $m_l = \max_i g_i^l$; $m_1 \ge m_2 \ge \cdots \ge m_l$ and $Z_l \ge m_1 + m_2 + \cdots + m_l$. Because $T \ge t$, $Z_T \ge Z_t \ge m_1 + m_2 + \cdots + m_t$. Hence, by equation (1), it suffices to show that $Z_T \ge t \max_{i=1}^t g_i^T = tm_T$. But this is clear because $Z_T \ge m_1 + m_2 + \cdots + m_t \ge tm_t \ge tm_T$. □

Here is a simple corollary, whose proof we omit.

COROLLARY 18 *For arbitrary nonnegative reals $h_1, h_2, ..., h_t$, if we start with $g_i^0 = h_i$ for all i, then however the game is played, by the time the adversary has made at least T moves of type 2, $C \ge G_0 = \sum_{i=1}^t h_i$, if $T \ge t$.*

LEMMA 19 *The number of iterations in which the algorithm chooses an interval which is disjoint from all of opt's t intervals is at most t.*

PROOF. Let I_i be the adversary's ith optimal interval, when sorted arbitrarily, $i = 1, ..., t$. Let Y_i be the portion of I_i which is left uncovered by the greedy algorithm at a given time; note that Y_i is an interval. The algorithm is greedy and could have chosen to add interval I_i. Adding I_i to the algorithm's intervals would have covered $length(Y_i)$ additional length; it follows that the increase in length of the greedily-covered region must be at least $\max_{i=1}^t length(Y_i)$.

C, initially 0, will denote the coverage of the greedy algorithm. g_i will denote the *fractionally-weighted* length of Y_i. Here we mean that a portion of Y_i that lies in some number m of the optimal intervals I_j is apportioned equally to those m I_j's. Clearly $g_i \le length(Y_i)$. Note that $\sum g_i$ is the length of the uncovered portion of $\cup_i I_i$.

When the algorithm chooses an interval J, either $J \cap Y_i$ has length 0 for all i, or it does not.

If $J \cap Y_i$ has length 0 for all i, then no Y_i will change in length. The coverage C must increase by at least $\max_i length(Y_i) \ge \max_i g_i$. (This is a type-2 move.)

If $J \cap Y_i$ has positive length for some i's, let δ_i be the fractionally weighted length of $J \cap Y_i$. The coverage increases (at least) in the region $J \cap (\cup_i Y_i)$, whose length is $\sum_i \delta_i$. C increases by at least $\sum_i \delta_i$. Each g_i decreases by exactly δ_i. (This is a type-1 move.)

Now let h_i be the fractionally weighted length of I_i. Note that $\sum h_i = length(\cup I_i) \ge L$, the target coverage (since the optimal algorithm is, of course, feasible). The corollary states that after t type-2 iterations, $C \ge \sum h_i \ge L$, and hence the greedy algorithm will terminate. □

LEMMA 20 *The algorithm chooses at most four intervals that contain the left endpoint of an opt interval I_i and at most four that contain the right endpoint of I_i.*

SKETCH. It suffices to prove the lemma for the left endpoint z of I_i. Assume the greedy algorithm chose five intervals containing z. Sort the five intervals by left endpoint. Use the fact that any sequence of five distinct integers has either an increasing subsequence of length three or a decreasing one of length three, and assume the former to infer that the algorithm chose three intervals, in chronological order, with increasing left endpoint. But now one sees that instead of choosing the second one, the algorithm should have chosen the third, since it would have increased coverage more. □

THEOREM 21 *The performance ratio of greedy is at most 9.*

SKETCH. Greedy can choose at most $8 = 2 \cdot 4$ intervals intersecting each of OPT's t intervals, and, by Lemma 19, at most t more. □