Framework and Algorithms for Trend Analysis in Massive Temporal Data Sets

Sreenivas Gollapudi
SUNY Buffalo

D. Sivakumar
IBM Almaden

ABSTRACT

Mining massive temporal data streams for significant trends, emerging buzz, and unusually high or low activity is an important problem with several commercial applications. In this paper, we propose a framework based on relational records and metric spaces to study such problems. Our framework provides the necessary mathematical underpinnings for this genre of problems, and leads to efficient algorithms in the stream/sort model of massive data sets (where the algorithm makes passes over the data, computes a new stream on the fly, and is allowed to sort the intermediate data). Our algorithm makes novel use of metric approximations in the data stream context, and highlights the role of hierarchical organization of large data sets in designing efficient algorithms in the stream/sort model.

1. INTRODUCTION

The problem of finding significant trends from massive data sets has many applications. Some of the features of these data sets that make this problem challenging are the facts that such data sets are large, distributed, and evolve with time. Prominent examples include analyzing point-of-sales information from various stores distributed geographically, extracting trends from inventory data, buzz analysis for products (e.g., music albums) where "impressions" are collected from various sources of information over long periods of time, correlation analysis between multiple temporal data streams arising, for example, from financial market transactions and news source references, and discovering trends from medical data over periods of time and across large geographies.

In this paper, we provide a rigorous framework and a family of algorithms for this class of problems. The data model we use is based on streams of relations, where each attribute is endowed with a hierarchically organized metric. The computational model in our framework includes primitives for streaming and sorting, together with a natural cost function for various operations. The confluence of our data and computational models enables us to address a rich variety of real-world trend analysis problems through a unified algorithmic core; we formulate and solve this core problem through a Markovian model.

What is a trend? Consider a large data set of tuples of the form (κ, s, t), where κ is the primary key attribute, s indicates a numeric field of interest, which we will call the *volume*, and t denotes time. The goal is to understand significant activity (rise/fall) in the volume field over specific intervals of time, *and* explain such activity by succinctly describing a subset of the primary key space that is responsible for it. For example, consider a grocery store sales database where records are of the form
$(\text{STOREID}, \text{PRODUCTID}, \text{SALEAMT}, \text{DATE})$,
where $(\text{STOREID}, \text{PRODUCTID})$ forms the primary key and SALEAMT is the volume key. Here the goal is to extract intervals of time where *large* volumes of sales occur in *small* and *compactly described* subsets of the primary key space.

Each of the three qualifiers used above deserves a few words of explanation. Firstly, if our main goal is to identify large volumes of sales, one can trivially report the entire primary key space, which is not very useful. Secondly, if our goal is to identify small subsets of primary key space, the most popular (in terms of sales) primary key value will solve the problem, but we don't expect any one key to account for more than a small fraction of the total volume. Finally, even if we manage to find a small subset of the key space that accounts for a fairly large fraction of the total volume in a time interval (e.g., the most popular key values that together account for 5% of all volume), if the subset of the key space lacks any compact description (and is simply an enumeration of seemingly unrelated key values), it is of little use to the analyst. In particular, if there is no theme that unifies these key values into a coherent subset with real-world semantics, it is unlikely to be of much value to the business, and in any case, certainly does not meet our intuitive notion of a trend.

Our work provides the first formal and simultaneous treatment of the three principal components in a framework for trend analysis. Namely, we address all three of the following questions:
— what is the data model?
— what are the allowed computational primitives, and what are their costs?
— what is the structure of the output?

We address the first question by adopting a simple relational model of data together with certain additional enhancements. For example, we will posit a metric structure on each attribute that participates in the primary key, and further organize this metric into a highly structured hierarchy. This model, to a large extent, is a reflection of common data schemas employed in various application domains; the metric enhancements we require have the property that they can be synthesized in an offline manner and supplied as auxiliary input to our algorithms.

Our computational primitives include basic streaming access to data, simple filters that read one tuple at a time and either discard it or write it back with some modifications, and sort/aggregate operations. Notice that the only global operation we perform on data streams is the sort/aggregate operation; this operation is highly optimized in every commercial database system, and there are very good arguments [2] why one may consider sorting as a fundamental enhancement to data stream algorithms. It will turn out that the interplay between our data and computational models leads to a natural cost function for various operations.

The output of trend analysis algorithms in our framework is a collection of focused regions of the primary key space together with time intervals where these regions exhibit unusually high or low activity in terms of volume. To describe these regions succinctly, we take advantage of the uniform and hierarchical decompositions of the primary key domains.

The core algorithmic task that emerges from our framework is the following. Given a collection of temporal data streams, identify well-supported bursts of activity in the *sum* of the given streams. By well-supported we mean a burst of activity that is accounted for by a small number of the input streams. We adopt a Markov analysis model for this problem, similar to [16], who used it in the context of discovering bursts in streams of information.

Related work. One of the recent works on trend analysis is Kleinberg's work mentioned above. Whereas Kleinberg's work focuses on trend discovery, we formulate and solve the more general trend analysis problem; our algorithmic component is inspired by [16]. The work of Lakshmanan et al. [17] on generalized minimum description length approach for summarizing data is perhaps the closest to ours; they also consider data organized as multiple hierarchies, and attempt to explain as much of the data as possible using succinctly described subsets of the attribute space. Our work has two additional features, namely, we study the temporal trend analysis problem rather than the static summarization problem, and secondly, we seek to *explain* important trends in data in addition to discovering their existence.

Aggarwal [1] offers a completely different framework for mining evolving data, in terms of spatial and temporal velocity profiles that admit convenient visualizations; our formulation of trends and their analyses are combinatorial in nature, and we work with a more refined computational model of streams. Ganti, Gehrke, and Ramakrishnan [11] study the problem of maintaining a data mining model under large scale evolutions of data streams. Domingos et al. [8, 15] (see also [9]) study algorithms for learning and maintaining decision trees from temporal streams. Yang and Widom [19] propose structures to support aggregate queries over temporal data. Beginning with the seminal work of [5], there is a considerable body of work in the pure streaming model for computing various statistical quantities [5, 10, 12, 13, 14]. In recent work, Aggarwal et al. [2] study several "extended" models of data stream computations, with particular emphasis on database-friendly models.

2. PROBLEM FORMULATION

2.1 Data sources, format, and distribution

Our basic data model comprises a single relational schema with $k+2$ attributes (for some $k \geq 1$). Here k attributes together constitute the primary key; of the other two, one indicates time (e.g., date), and the other contains the main quantity of interest whose trends (over time) we wish to understand. Formally, we consider a $(k+2)$-ary relation $\mathcal{R} \subseteq S \times T \times D_1 \times \ldots \times D_k$, where, wlog. $S = \mathbf{R}$, $T = \mathbf{Z}$, and each D_i is a finite set. The trailing k attributes together constitute a primary key for the relation. Let $\mathcal{D} = D_1 \times \ldots \times D_k$ where d denote the size of the largest D_i. The first attribute S denotes the *volume dimension* while the second attribute T represents the *temporal dimension*.

The relation $\mathcal{R}$ could be distributed across several data sources (as is commonly the case, e.g., sales records), but all sources have records of the same schema. Typically, one of the last k fields will identify which source each tuple comes from.

Example 1. *A typical input relation for a grocery store chain trend analysis problem consists of tuples of the form* (SALEAMT, SALEDATE, STOREID, PRODUCTID)*, and the goal is to understand trends in the field* SALEAMT *in terms of the* STOREID *and* SALEAMT *fields.*

To support large scale analysis, it is important to understand what operations admit efficient implementation in this model. We will take the view that the following three operations are natural and rich enough to support a variety of analysis algorithms, and at the same time, simple enough to be applied to a large distributed data set.

(1) *Read-only streaming.* Read in the tuples of a large distributed relation with N tuples as a stream, and process each tuple quickly (e.g., in time polylogarithmic in N), using very little internal memory (also polylogarithmic in N).

(2) *Read-write streaming.* Read in the tuples of a large distributed relation with N tuples as a stream, process each tuple quickly, and *write* a (small) number of tuples (possibly zero) into an output stream. Again, the internal memory used for the streaming process should be very small.

(3) *Sorting and aggregation.* Sort an intermediate stream on one or more of the attributes, and optionally, aggregate all tuples with identical values in some attribute into a single tuple (e.g., sum the first attribute over all tuples with the same value in the third attribute).

There are two related aspects of the model that are worth highlighting. One is the number of intermediate streams that are active at any point of the computation; we require that this should be $O(1)$, independent of the number of tuples in the stream. The second aspect is that even though we consider a distributed data set to be a single stream, it is clearly advantageous to be able to do the processing of these streams independently, without having to process the first part before moving on to the second part. This, of course, depends on the exact processing being done. In our algorithm to be described later, this is indeed the case, namely the streams can often be acted upon independently. In some cases, we do need the data set to be processed as one stream, but this typically happens on an "aggregated intermediate stream." Since the aggregated intermediate streams that we produce will be smaller than the input streams by several orders of magnitude, they can brought into a single storage system, often in-memory, and processed locally.

2.2 Trends

Intuitively, a *trend* is some temporal feature of the data set that is distinguished in some respect, and that can be succinctly summarized in a sentence or two. Formally, we define a trend as a succinct combinatorial rectangle, defined below.

DEFINITION 1. *(**Combinatorial rectangle**) A combinatorial rectangle A in $\mathcal{D} = D_1 \times D_2 \times \ldots \times D_k$ is a subset of the form $A_1 \times \ldots \times A_k$, where $A_i \subseteq D_i$ for each i.*

A combinatorial rectangle $A_1 \times \ldots \times A_k$ could be as large as $\Omega(d^k)$; the standard way to describe the rectangle is to enumerate the sets A_j. Note that the size of this standard description is $|A_1|+|A_2|+\ldots+|A_k|$, which could be as large as $\Omega(\max\{|D_1|,\ldots,|D_k|\}) = \Omega(d)$. Recall that $d \gg k$.

DEFINITION 2. *(**Succinct combinatorial rectangle**) A* combinatorial rectangle with succinct description *is a combinatorial rectangle in $\mathcal{D} = D_1 \times \ldots \times D_k$ whose description is of size $O(k)$.*

DEFINITION 3. *(**Trend**) We define a* trend *to be a pair of the form (A, I), where A is a combinatorial rectangle in $\mathcal{D}$ with succinct description and I is a time interval.*

Our goal will be to discover regions in the attribute space and time intervals where these regions exhibit significant and/or unusual activity. In a sense, this is similar to OLAP cube analyses, but our use of tree metrics (to be described) makes it rather different, and enables the discovery of "automatically inferred" rectangles, and not necessarily rectangles of "continuous subsets" of the attribute space.

In the grocery chain trend analysis example, we would like to make conclusions of the form "sales is up 5% from dairy products and poultry in the northwest region." To do so, one needs to cluster products and geographic regions into groups, explicitly taking into account similarities between various clusters. One of the key contributions of our work is that such clustering is placed into the pre-processing step through the use of suitable metric spaces and approximations of metric spaces by *tree metrics*. The application of tree metrics to automatically factor out such similarities is presented in the following sections.

2.3 Metrics on attribute spaces

DEFINITION 4. *A* metric *on an attribute space D is a mapping $\mu : D^2 \longrightarrow \mathbf{R}$ that satisfies the following conditions for all $x, y, z \in D$.*
(Non-negativity) $\mu(x,y) \geq 0$ and $\mu(x,y) = 0$ iff $x = y$;
(Symmetry) $\mu(x,y) = \mu(y,x)$
(Triangle inequality) $\mu(x,y) \leq \mu(x,z) + \mu(z,y)$

We assume that for each i, $1 \leq i \leq k$, we have a metric μ_i on the set D_i such that computing $\mu_i(x,y)$ for any $x, y \in D_i$ is a unit cost operation. The idea of using metrics on each of the D_i's is to enable trend discoveries to take into account similarities between various values of each attribute.

Example 2. In our grocery chain example, the metric on "STOREID" captures various forms of similarity between stores, including geographic, demographic, store size, location of competitors nearby, etc; the metric on "PRODUCTID" captures similarity between products (e.g., potatoes and cauliflowers are more similar than are potatoes and shampoos, etc.); the metric on clinical symptoms in medical data captures various forms of similarity between the symptoms.

An important feature of our framework is that these metrics are assumed to be synthesized in the pre-processing step, and do not contribute to any extra cost during the analysis step. Furthermore, we do not assume any metric on $D_1 \times \ldots \times D_k$. There are several reasons for this: (1) computational intractability of explicitly creating and processing a metric on the product space; (2) more importantly, working with a product metric space forces the user to find a way to combine metrics on completely unrelated domains, e.g., zipcodes and product categories in grocery store sales. Instead, we will obtain a hierarchical clustering of each of the individual metrics by approximating it by a tree metric, and work with various products of the trees used in the approximation.

Note that the setup outlined above achieves a nice tradeoff. On the one hand, we do not require the user to artificially make the entities in various attributes comparable (so that one could use, e.g., principal component analyses and other spectral techniques to identify features that are linear combinations of individual attributes). On the other hand, within each attribute, we allow the user considerable latitude in defining similarity between various values. This is a useful feature since, e.g., a grocery store in a suburban California city might be quite similar to one in suburban Nebraska based on store size, demographics, etc., even though they are geographically far. Similarly, similarity between products could be (pre-)computed based, e.g., on how often two products have been sold to the same customer in the same transaction (thus we do not preclude beer and diaper, to name a popular example, from being deemed similar).

2.4 Tree metrics

The next crucial idea, one that is especially important from the viewpoint of producing *succinct* descriptions of trends, is that of hierarchically organizing each metric. The underlying motivation for this is the following. In temporal data analysis and visualization, an extremely useful fact is that time is a "compressible" axis that can be treated at various "resolutions." This allows us, for example, to plot daily stock closing prices for ten years within a small window.

The reason for imposing a metric structure on the dimensions of the attribute space is to enable such multiresolution analysis of categorical data. If each of the k metrics $\mu_1, \ldots, \mu_k$ naturally admits a hierarchical decomposition, then we could use these decompositions to achieve this ability. Even if a metric does not admit a natural decomposition, the powerful theorem of Bartal [6, 7] allows us to probabilistically approximate *any* metric by a *tree metric*, in fact, a metric of "hierarchically well-separated" trees. Informally, a tree metric on a space D is a edge-weighted tree whose leaves are the points of D, and where the distance between two points $x, y \in D$ is given by the sum of edge weights between the (unique) path from x to y in the tree. By sacrificing a small ($= O(\log n \log\log n)$) factor in the quality of approximation of the metric, we will assume that each metric is a tree metric, and furthermore, starting from the root and walking down to a leaf, each edge is twice (say) as heavy as the next edge. See Appendix for a summary of Bartal's tree approximations and formal definitions of related concepts.

The idea of using tree metrics in our context is rather powerful, since it allows us to study the product space $D_1 \times \ldots \times D_k$ in a principled manner, retaining the metric properties of each D_i, without having to forcibly create a metric on the product space. Thus each attribute now is in a metric space, *and* while the values are incomparable across metric spaces, the metric spaces are structurally quite similar (hierarchically separated trees). Furthermore, inside each metric space, a tree metric approximation has the advantage that similar values are naturally grouped and the correlations are automatically organized meaningfully. Note that in a tree metric, at any node, the children of the node are all "equally dissimilar"—this is a feature that is usually absent in typical hierarchical organization of metrics. This relieves the algorithm from having to explicitly account for the similarities between various values in each of the metric spaces. In other words, we are able to design algorithms that are completely oblivious to the fine details of each metric, and that are able to treat all of the metrics in a uniform manner.

We stress the fact that the tree metric approximation theorem of Bartal offers us a way to treat all metric spaces in a systematic manner, by treating each metric as a hierarchy of the uniform metric spaces. The field of hierarchically clustering metric spaces has a long and rich history, and in some applications, it is possible that one can derive hierarchical decompositions based on the structure of the domain and the context of the application (e.g., see the work of Aggarwal et al. [3]). In such cases, one might wish to use the tree metric obtained by other methods in lieu of applying Bartal's probabilistic approximation by tree metrics. In any case, we will assume for the rest of the paper, that each metric μ_i is a tree metric, and we will denote the tree for μ_i (on D_i) by U_i.

Definition 5 (Trends from Tree metrics). *Let T denote the time interval of interest. If, for $1 \le i \le k$, μ_i is a tree metric on D_i whose tree is denoted by U_i, a* trend *in $\mathcal{D} = D_1 \times \ldots \times D_k$ is a collection of tuples of the form $((v_1, \ldots, v_k), I)$, where each v_i is a node in the tree U_i, and $I \subseteq T$.*

Proposition 1. *Let $U_1, \ldots, U_k$ be trees such that the leaves of U_i are all the points of D_i. If $v_1, \ldots, v_k$ are nodes, respectively, in trees $U_1, \ldots, U_k$, the set of tuples $\{(x_1, \ldots, x_k)$, where each x_i is a leaf in the sub-tree of v_i, is a combinatorial rectangle. Moreover, any trend corresponding to this rectangle has an $O(k)$ size description, in the form of $(v_1, \ldots, v_k)$.*

Note also that the subtree under v_i is a collection of points in D_i with bounded diameter, and we can read off the bound from the tree metric μ_i. This allows the user to make queries of the form "give me trends where the diameter on the geographic axis is 50 miles, the diameter on the product categories is 10, etc."

A trend corresponding to a node $(v_1, \ldots, v_k)$ in $U_1 \times \ldots \times U_k$ has the following property: if most of the v_i's are nodes that have large sub-trees under them, then the trend is rather "large" in the sense that it applies to a large part of the product space $\mathcal{D}$; similarly, a trend $(v_1, \ldots, v_k)$ where most of the v_i's are nodes with small sub-trees corresponds to a "focused trend" that is restricted to a rather fine portion of the input tuple space. One may expect that the former type of trends are able to account for much of the significant activity in the volume axis, but are often rather not very useful. The latter type of trends are more likely to account for only small activity in the volume axis, but are more likely to be "actionable," namely they yield, from a business perspective, more useful information. An example of a large trend is "produce sales in California in the period July–September was 20% more," while an example of a focused trend is "frozen vegetable sales was 3% lower in California cities with population less than 500K during the last two weeks of January." Clearly, in the latter case, the grocery store chain could offer promotional sales for frozen vegetable in the appropriate market segment, and also explore other underlying reasons for the phenomenon.

When a trend analysis module outputs a collection of trends, it is very important that several trends do not refer to the same subspace of the product space $\mathcal{D} \times T$ — in other words, different trends in the collection should highlight different parts of the space $\mathcal{D} \times T$. We capture this idea in the following definition.

Definition 6 (Non-subsumptive trends). *A collection of trends $\{(v^{(j)}, I^{(j)})\}$ is said to be* subsumptive *if for some pair j, l, $I^{(j)} \cap I^{(l)} \neq \emptyset$ and for every i, $1 \le i \le k$, either $v_i^{(j)}$ is an ancestor of $v_i^{(l)}$ or $v_i^{(l)}$ is an ancestor of $v_i^{(j)}$. The collection is said to be* non-subsumptive *if no such pair exists.*

In other words, two trends with overlapping intervals should be ruled out if, in each tree, one of the nodes is a "finer refinement" (descendant) of the other. For example, it is obvious that we wish to rule out having both trends
(west, produce, july) and
(calif, cauliflower, july).
A somewhat subtle case is having both
(west, cauliflower, july) and
(calif, produce, july).
Since the sales of cauliflower in California during the month of July contributes to the volume under both nodes, we will exclude them from both being present in the output of the trend analyzer. To pick which of the two is more significant, we remove the common part, namely cauliflower sales in California, from both nodes and choose the one that has higher remaining volume.

Our algorithm will always produce a collection of non-subsumptive trends.

2.5 Computational costs

The foregoing discussion naturally highlights another, more computational, aspect of the trend analysis problem. Namely, to produce finer snippets of intelligence, it is natural to expect an algorithm to be computationally more expensive. Indeed, this is the case, and the hierarchical organization of data yields a natural cost model that brings out this fact. Recall that in our model, the basic operations involve translating a stream of data into a more compact stream by processing, sorting, and aggregating. It will turn out that our algorithm will "operate" at some node of the product of the trees $U_1, \ldots, U_k$, and each node v of this tree corresponds to a node v_1 in U_1, a node v_2 in U_2, and so on. Our algorithm will analyze the nodes in a top-down fashion; to analyze the node v, we will stream through the data, retain only those records whose values fall under the sub-trees of $v_1, \ldots, v_k$, then further aggregate the values up to the nodes $v_1, \ldots, v_k$. This results in a single temporal stream that corresponds to the node v, which will then be analyzed. Notice that the key high-level operations we perform in this style of analysis is that of "collapsing" data to some node $v = (v_1, \ldots, v_k)$ by a stream/sort/aggregate process. The higher the nodes $v_1, \ldots, v_k$ are in their trees, the more records we expect to process, and the sort operations are likely to be costlier (especially if the data is distributed). Thus there is a natural cost associated with the decision to explore any given node of the product of the trees.

The stream/sort model has recently been proposed [2] as a natural computational model that takes into account the various capabilities in modern database systems. Our setup naturally benefits from this model, and also offers further justification for the usefulness of this model. The success of our application is achieved through the confluence of the various factors, specifically, the three basic operations outlined above, and the high-level operation of "collapsing" data to some node in a product of k tree metrics. We anticipate that these ideas will be valuable algorithmic paradigms for very large data sets.

3. MODULES FOR TREND ANALYSIS

We now present some modules that will be used in our trend analysis algorithm. The first module is the setup module, and a the second one is the trend discovery module. corresponds to seeking explanations of trends discovered in terms of sub-domains of $\mathcal{D}$.

3.1 Setup module

The input to the setup module consists of a node $v = (v_1, \ldots, v_k)$ in the product tree $U = U_1 \times \ldots \times U_k$, an attribute i, $1 \leq i \leq k$, and a time interval $I = [t_s, t_f]$. Note that for each j, $1 \leq j \leq k$, v_j is a node of U_j. Suppose the node v_i in U_i has l children, labeled $v_i^1, \ldots, v_i^l$. For $1 \leq m \leq l$, $w_j = (v_1, \ldots, v_{i-1}, v_i^m, v_{i+1}, \ldots, v_k)$ denote the m-th child of v in the i-th attribute.

The task of the setup phase, given (v, I), is to produce streams $R, R_1, \ldots, R_l$ that correspond to the sub-domain of $\mathcal{D}$, respectively, under $v, w_1, \ldots, w_l$, and the time interval $I \subseteq T$. These streams will be used for further trend discovery and analysis.

Consider the trend analysis problem for a grocery store chain with $k = 2$. Let us suppose that the first attribute corresponds to geographic location and the second attribute corresponds to category of the product sold. The quantity of interest is sales. Let v denote the node corresponding to (WEST, PRODUCE), where WEST stands for the Western U.S. Suppose the interval $[t_s, t_f]$ corresponds to the month September. If $i = 1$, we are asking for an analysis in the geographic attribute, namely we wish to find unusually high or low activity in sales that has an explanation in terms of geography, for example a conclusion of the form "The change in sales during September comes mainly from Colorado and Idaho in the period 9/14–9/22." If $i = 2$, we ask for a trend with an explanation in terms of the product category, for example, "The change in sales during September comes mainly from leafy vegetables during the period 9/15–9/23." Here it is important to keep in mind that our tree nodes may not have completely natural semantics like "PRODUCE", "WEST", "Colorado", etc; if Colorado and Idaho are deemed to be similar by the metric used, they could very well be part of the same subtree.

The goal is to identify the most important trends in the time period $[t_s, t_f]$ from the portion of the input data stream that satisfies the following conditions. A tuple $(s, t, a_1, \ldots, a_k)$ is included if and only if (1) $t \in [t_s, t_f]$, and (2) for each j, $1 \leq j \leq k$, the value $a_j \in D_j$ is in the sub-tree under the node v_j in the tree U_j that corresponds to the metric μ_j on D_j.

The relevant tuples are obtained by the following stream/sort process. When a tuple $(s, t, a_1, \ldots, a_k)$ is processed, first we check if it satisfies the above criteria. If it does, then we know that a_i is in the sub-tree of v_i in U_i; suppose a_i is, in fact, in the sub-tree under v_i^m, the m-th child of v_i. Then we produce the tuple (s, t, m) in the intermediate data stream. This stream is then sorted using the second attribute as the primary sort key and the third attribute as the secondary sort key; thus, if $t < t'$, all tuples of the form $(\cdot, t, \cdot)$ will precede all tuples of the form $(\cdot, t', \cdot)$, and for each t, if $m < m'$, then all tuples of the form $(\cdot, t, m)$ will precede $(\cdot, t, m')$. This stream is then aggregated on the third attribute, namely the collection of tuples of the form $(\cdot, t, m)$ are replaced by a single tuple whose first field is the sum of the first fields of such tuples. Thus we obtain a new temporal data stream with tuples of the form (s, t, m), where $t \in [t_s, t_f]$ and for each t and m, there is at most one tuple of this form.

For $1 \leq m \leq l$, let R_m denote the collection of all such tuples where the value of the third attribute is m; this stream is naturally associated with the m-th child w_m of v. If we further replace all tuples of the form $(\cdot, t, \cdot)$ by a single tuple (s, t) whose first field is the sum of the first fields of such tuples, we obtain a temporal data stream R that corresponds to the node v in U.

In our example, suppose WEST has three children, NW, CALIF, SW, and PRODUCE has two children, FRUITS and VEG. If $i = 1$, namely we are analyzing in the geographic attribute, then we produce a temporal stream for each of (WEST, PRODUCE), (NW, PRODUCE), (CALIF, PRODUCE), and (SW, PRODUCE); in the stream for (CALIF, PRODUCE), for example, for each date t, there is one tuple (s, t), where s denotes the total sales from California on date t (over all produce). Similarly, if $i = 2$, we produce a stream for each of (WEST, PRODUCE), (WEST, FRUITS), and (WEST, VEG), and the stream for (WEST, FRUITS) contains, for each t, one tuple of the form (s, t), where s denotes the total sales of fruits, summed over all the states in the west.

To summarize, by simple filtering, streaming and sorting operations, we produce $l + 1$ streams $R, R_1, \ldots, R_l$, where R is the temporal data stream associated with the node v, and the stream R_m is associated with the m-th child w_m of v. All of the $l + 1$ streams lie in the time interval $[t_s, t_f]$. For $1 \leq m \leq l$ and $t \in [t_s, t_f]$, we will denote by $R_m(t)$ the unique s such that the tuple (s, t) belongs to the stream R_m. Note also that for any time t, $R(t) = \sum_{m=1}^{l} R_m(t)$. Finally, note that since the only operations on the raw data involves streaming, filtering, and sorting, this phase easily admits distributed implementation similar to mergesort.

3.2 Trend discovery module

The input to the trend analysis module consists of node

v and interval I, together with a "volume threshold" v_{min}. The output of the trend analysis module consists of a collection of non-overlapping intervals $I_1, \ldots, I_r$ that correspond to "bursts" of activity in the stream R corresponding to (v, I), whose volume is at least v_{min}.

The goal of this phase is to mine the stream $R = R(v, I)$ produced as outlined above, and to detect significant trends from this stream. Here we use a type of hidden Markov analysis, similar to Kleinberg's use [16] in detecting bursts in information streams. In the generic setup we use, we assume that there is a finite or infinite Markov chain $\mathcal{M}$ with state set Q, value set B, and for each $q \in Q$, there is a probability density function $P_q : B \longrightarrow \mathbf{R}$, and for states $q, q' \in Q$, there is a cost $C_\tau(q, q')$, with the following semantics. From any state, the chain emits a symbol, and transitions to another state. If the chain is in state q, the probability that it emits the value b is $P_q(b)$. Similarly, the cost of transitioning from state q to q' is $C_\tau(q, q')$; we further assume $C_\tau(q, q) = 0$ for all $q \in Q$. The problem is, given a sequence of values $(b_1, \ldots, b_T)$, the goal is to find the sequence $(q_1, \ldots, q_T)$ of states that optimizes the following two criteria:

(1) the probability $\prod_i P_{q_i}(b_i)$ is maximized;

(2) the total transition cost $\sum_{1 \le i < T} C_\tau(q_i, q_{i+1})$ is minimized.

Condition (1) states that the state sequence $(q_1, \ldots, q_T)$ explains the observed data as best as possible; however, note that to do this, one may simply select, for each i, the state whose likelihood of emitting b_i is maximum. Our notion of a trend intuitively has a more "continuous" ascend or descend, and condition (2) captures this by implicitly restricting the number of state transitions.

In our algorithmic framework, we employ this paradigm to define a Markov chain $\mathcal{M}$ that is used to identify the most important bursts of activity in the data stream R. The Markov chain $\mathcal{M}$ consists of states that correspond to successively higher volumes; namely, we take $B = \mathbf{R}$ and for some $\epsilon > 0$ the probability density function for state q is given by $P_q(b) = \beta e^{-b/\beta}$, where $\beta = (1+\epsilon)^{-q}$; note that the expected value of this distribution is $(1+\epsilon)^q$. The transition cost function C_τ for $\mathcal{M}_1$ has the property that $C_\tau(q, q')$ is proportional to $|q - q'|$. To turn the bi-criteria optimization into a tractable computational problem, it is customary to attach weights to each of the criteria, and to optimize a single cost function. For example, we will use the formulation of minimizing the following cost:

$$\sum_{t=1}^{T} (\alpha_1 C_e(q_t, b_t) + \alpha_2 C_\tau(q_t, q_{t+1})), \tag{1}$$

where $C_e(q, b) = -\ln P_q(b)$ is the cost of "explaining" symbol b by state q, and $q_{T+1} = q_T$, so $C_\tau(q_T, q_{T+1}) = 0$. We define the transition cost $C_\tau(r, q)$ to be $C_e(r, (1+\epsilon)^q) - C_e(q, (1+\epsilon)^q)$, that is, the cost of transitioning from state r to state q is equal to the additional cost incurred by explaining the average volume for state q using state r.

Here, the parameters α_1 and α_2 help us balance the contributions of the two terms, and allow us a trade-off between accurate explanation of data and the smoothness of the state sequence. Computing the state sequence that minimizes this cost function can be done by standard dynamic programming techniques. We use this chain to compute a state sequence that minimizes the cost function of the form described above. Thus, the rate at which a state generates "volume" depends exponentially on the index of the state. Once the optimal state sequence has been computed, a *burst of intensity* q corresponds to all time intervals where the state is q or higher (similar to [16]). By employing $\mathcal{M}_1$ with appropriate parameters, we identify the most important bursts of activity in the stream R in the given time period $[t_s, t_f]$ using the optimization problem on $\mathcal{M}_1$.

To produce a sequence of non-overlapping bursts of volume at least v_{min}, we first discard all bursts that do not satisfy the volume constraint; next, we sort the bursts according to a score (to be described shortly), and greedily pick non-overlapping bursts, starting with the burst of highest score. The score for a burst in interval J will be defined as the ratio of the average volume in J to the average volume in the smallest interval J' such that $J \subseteq J'$ (taking $J' = I$ for the largest interval J). This score achieves a nice-tradeoff between narrow bursts of very high intensity and wider bursts of moderate intensity.

Implementation Notes. In terms of implementation, there is a straightforward dynamic programming algorithms to compute the state sequence of minimum total cost for a given sequence of values. Namely, if the given sequence is $b_1, \ldots, b_T$ and the chain has N states, then define an $N \times T$ cost matrix C, where $C(q, t)$ is the minimum cost, among all state sequences that end in q, to output the sequence $b_1, \ldots, b_t$ of values. It is easy to see that

$$C(q, t) = \alpha_1 C_e(q, b_t) + \alpha_2 \min_r (C(r, t) + C_\tau(r, q)),$$

with suitable boundary conditions on $C(\cdot, 0)$. The boundary conditions for $t = 0$ depend on where we would like to start the Markov chain; some natural choices are to start it at the state q^* where C_e is lowest, thus $C(q^*, 0) = 0$ and $C(q, 0) = \infty$ for all states $q \neq q^*$.

We summarize the computational efficiency of the basic module in the following theorem.

THEOREM 1. *The basic module, invoked with node $v = (v_1, \ldots, v_k)$, interval $I = [t_s, t_f]$, and attribute i, can be accomplished in four streaming passes. The first pass is a read-write streaming/filtering pass where only those tuples whose values are in the subtrees of $v_1, \ldots, v_k$, and whose time field is in I, are retained; the second pass is a sort/aggregate pass that aggregates the tuples to the levels of the nodes $v_1, \ldots, v_k$; the third pass is a read-only streaming pass during which the cost matrix C_1 needed for the optimal state sequence problem for the Markov chain $\mathcal{M}_1$ is computed at node v_i, and the columns $C_1(\cdot, t)$ is written out for each time step $t \in I$; the fourth pass is a read-only streaming pass that will reconstruct the optimal state sequence of $\mathcal{M}_1$.* $\square$

Note that if the time interval I is sufficiently small (e.g., a year, measured in days), all the required computations could be accomplished in memory without any sort operations; this leads to further additional savings. In general, since time is a "compressible axis," it should always be possible to run the algorithm in this fashion; for example, in our experiments, we were able to handle 10 years worth of stock data in memory relatively easily.

4. TREND ANALYSIS ALGORITHM

In this section, we build on the module introduced in Section 3, and present the complete algorithm for trend analysis. In this formulation, we require the user to specify a volume constraint on the trends the algorithm is supposed to discover.

The *volume constraint* specifies a real number $v_{min} \in (0,1]$; this constraint has the interpretation that any trend $(v = (v_1, \ldots, v_k), I)$ output by the algorithm must account for at least v_{min} fraction of the total volume during interval I. The intent of this constraint is that the trends discovered by the algorithm have non-trivial significance in terms of volume. A refinement of this constraint is to require that trend (v, I) must account for at least $v_{min}\eta^d$ fraction of the total volume during interval I, where η is a constant less than 1, and $d = \max_{1 \le i \le k} \mathrm{depth}(v_i)$. In other words, for deeper nodes (that correspond to smaller combinatorial rectangles), the volume constraint becomes relaxed.

Another alternative is to use *diameter constraints* in each of the k attributes. The i-th such constraint, $1 \le i \le k$, specifies a real number $\delta_i > 0$, and has the interpretation that any trend $(v_1, \ldots, v_k)$ output by the algorithm must satisfy $diam(v_i) \le \delta_i$, where $diam(v_i)$ denotes the diameter of the set of points under the sub-tree rooted at v_i in the metric tree U_i. The intent of these constraints is that the trends discovered by the algorithm are sufficiently focused in each of the k attributes. In this paper, our description of the algorithm and the experiments employ the volume constraint. Note that it is easy to satisfy either the first constraint at the expense of the other k constraints (just output the root), or the second set of k constraints at the expense of the first (just output the leaves).

The complete trend analysis algorithm is presented below. In the following, we set $\eta = 2/3$.

```
Trend-Analyzer($v = (v_1, \ldots, v_k), I = [t_s, t_f], v_{min}$)

(0) For $1 \le i \le k$, let $l_i$ denote number of children of $v$ in
    dimension $i$; let $w^i_1, \ldots, w^i_{l_i}$ denote these children.

(1) Let $R$ denote the stream corresponding to $(v, I)$, and
    for $1 \le i \le k$, $1 \le m \le l_i$, let $R^i_m$ denote the stream
    corresponding to $w^i_m$. Produce these streams using the
    setup module.

(2) Invoke trend discovery module on $R = R(v, I)$ and
    threshold $v_{min}$. Let $B$ denote the set of bursts produced.

(3) For each $b \in B$ do:

    (3.1) Let $J$ denote the interval of burst $b$.
    (3.2) For each dimension $i$, $1 \le i \le k$, and for each child
          $w^i_m$ in dimension $i$, $1 \le m \le l_i$, let $\theta^i_m$ denote the
          volume of $R^i_m$ in interval $J$.
    (3.3) Let $i^* = \arg\max_i \max_m \theta^i_m$, that is, some child in
          the $i$-th dimension has the largest volume within
          interval $J$, among all children of $v$ across all dimensions.
    (3.4) For each $m$, $1 \le m \le l_{i^*}$, if $\theta^{i^*}_m \ge v_{min}$, then
          recursively call Trend-Analyzer($w^i_m, J, \eta v_{min}$),
          and collect the trends reported.
    (3.5) If the recursive calls do not produce any trends,
          then report the trend $b$ for node $v$ and interval $I$.
```

Remarks.

(1) Note that the algorithm outlined above is independent of the implementation of the basic module, in particular, the parameters of the Markov chain $\mathcal{M}_1$. Thus we have managed to achieve a very degree of abstraction from the input data domains (via tree metrics), similarities among values of each attribute (via metric spaces), and also from the exact details of how the basic module works. For example, it is entirely conceivable that one uses sampling based algorithms for the basic module, that is, to detect the most important children of v in the i-th attribute during interval $[t_s, t_f]$.

(2) Since we used hierarchically separated trees in approximating our metric spaces, the diameter of the sub-tree falls by a constant factor whenever we go from a node to any of its children. This implies that, if we use diameter constraints, they are likely to be satisfied in a small number of iterations.

(3) The exact parameters of the Markov chain will dictate one of the most important computational criteria, namely the number of active nodes in L. Each parameter of the Markov chain has a natural interpretation, and thus they give the user a set of control variables with which to govern the quality/time trade-off in the execution of the algorithm.

5. EXPERIMENTS

In this section, we outline our experiments using on our algorithms for trend analysis. The goal of the experiments were the following:

(1) Obtain an efficient prototype of the algorithmic framework outlined.

(2) Demonstrate the efficacy of the trend analysis paradigm we have developed on non-trivially large real-world data, where readers can easily judge the quality of the output, and also verify it independently through standard data sources.

(3) Demonstrate the usefulness of tree metrics as very good ways to approximate naturally-defined metrics on attribute spaces, from the viewpoint of trend analysis.

(4) Demonstrate the efficacy of the trend analysis paradigm on massive multi-dimensional synthetic data.

5.1 Data sources

We consider two sets of data, the first one from financial markets, and the second one from a synthetic "basket data" generator that we have developed.

Financial data.

The financial data stream that we chose for our analysis the Standard & Poor's index S&P 500 ([18]) of stocks traded in the NYSE, AMEX, and NASDAQ stock markets, over the last ten years (March 1994 – Feb 2004). The analysis problem is to understand which of the 500 components of the index and subsets thereof are responsible for the various bursts of rise and fall in the S&P 500 index. In this instance, the 500 stocks that are included in the S&P 500 index are naturally organized into a tree based on the sectors, industries, etc. that the companies fall under. S&P 500 is a "market capitalization weighted" index, meaning that the index is a weighted sum of the stocks at the leaf level. For our purposes, we chose the 428 of the 500 stocks in S&P 500 that were traded on at least 96% of the time period considered, and the tree consisting only of these leaf nodes. While the weights that determine the S&P 500 change over time depending on the market capitalization of the various companies, we used a fixed weight scheme, depending on the market capitalization at a randomly chosen trading day within the period. Thus our "S&P 428" index is not exactly the S&P 500 index, but its rise and fall are identical to that of S&P 500 (figures omitted due to lack of space). We will denote the S&P tree, restricted to these stocks, by NATIVE-TREE-SP428.

The tuples in our relation were of the form
⟨CLOSING-PRICE, DATE, TICKER⟩,
where the TICKER field gives the ticker symbol for the stock (eg., IBM, ORCL, MSFT, etc.). The "volume" field (cf. Section 2.1) for our relation is the first attribute, namely the closing price, which refers to the weight-adjusted closing price of the stock[1]. By summing the CLOSING-PRICE fields of the children, we obtain a temporal stream for each of the internal nodes in the S&P 428 tree.

Synthetic basket data. Basket data (that arises from various shoppers buying collections of items at retail stores) is a standard benchmark in studying data mining algorithms; a standard basket data generator is given in the seminal work of Agrawal and Srikant [4], and subsequent extensions to temporal generators have also been studied []. We present a temporal basket data set generator that is suitable for multiple taxonomies.

The generator is based on a random walk on the product of trees corresponding to the taxonomies. Suppose we are given k trees $U_1, \ldots, U_k$, and a time interval T. The following real numbers will be used in the random walk, and their roles will become clear shortly: $j_1, \ldots, j_k, f_1, \ldots, f_k, \delta \in (0, 1)$. Pick a random leaf $v_i \in U_i$, for each i, and a random time step $t \in T$. Output the tuple $(v_1, \ldots, v_k, t, \theta)$, where θ is a randomly generated volume produced using a power-law distribution (so that most volumes are small, but there is a non-trivial number of large volumes). With probability δ, replace t by a randomly chosen value in T, and with prob. $(1-\delta)/2$, replace t by $t-1$ and with prob. $(1-\delta)/2$, replace t by $t+1$. For each i, $1 \le i \le k$, apply the following process. Replace v_i by a random leaf of U_i with probability j_i; with probability $1 - j_i$, do the following.

```
Let $v := v_i$.
Repeat:
    w.p. $f_i$, $v_i :=$ parent of $v_i$;
    w.p. $1 - f_i$, if $v_i$ is not a leaf, $v_i :=$ random child of $v_i$;
until $v_i$ is a leaf.
```

It is not hard to see that this random walk creates a number of "clusters" in the cross product of the sets of leaves of the trees and the time interval T.

We implemented this generator, and produced a data set of 3 trees, respectively, with 50, 100, and 150 leaves, and a million "sales" records with an average volume of 3, over a time period of one year.

5.2 Trees via metric approximation

One of the cornerstones of our algorithmic framework is the use of tree metrics to hierarchically cluster the underlying attribute space. While the S&P stock data is naturally organized into a hierarchy, to study the effectiveness of tree metric approximation, we implemented Bartal's metric decomposition algorithm to obtain a new tree metric on the S&P 428 stock data. To do this, we first defined a metric on the set of 428 stocks as follows.

DEFINITION 7 (BARTAL-TREE-SP428). *For a stock s, let $C_s(t)$ denote the closing price of s on trading day t. For a stock s, define vector $v_s \in \{-1, 0, +1\}^{2500}$, by*

$$v_s(t) = \begin{cases} -1 & \text{if } C_s(t) < C_s(t-1) \\ 0 & \text{if } C_s(t) = C_s(t-1) \\ +1 & \text{if } C_s(t) > C_s(t-1). \end{cases}$$

The 428×428 matrix H is defined by

$$H(s, s') = \sum_t |v_s(t) - v_{s'}(t)|.$$

For a stock s, let F_s denote the set of 20 stocks s' with the lowest values of $H(s, s')$. The metric μ on the set of 428 stocks is defined by

$$\mu(s, s') = \exp(|F_s \Delta F_{s'}|),$$

where Δ denotes the symmetric difference between sets. The tree obtained by Bartal's algorithm applied to μ will be denoted by BARTAL-TREE-SP428. □

The reason for not using H directly as a metric is that it is not robust: most of the distances are clustered in a rather short interval, and the diameter is very small. The metric μ, on the other hand, turns out to give excellent approximations to the S&P 500 tree, in the sense that the distance between stocks within shallow sub-trees is quite small, and the distance between stocks under very different sub-trees is quite large.

5.3 Results and Analysis

We now present some sample results obtained by applying our trend analysis.

Financial data.

The results for the native tree are given in Table 1, and the results for the tree derived using Bartal's metric approximations are given in Table 2. In the latter, the "node names" were assinged by us by picking the most significant node in the native tree that corresponds to the leaves in the trend discovered by our algorithm. By suitably adjusting the volume constraint and/or the parameters of the Markov chain, we were able to obtain various interesting trends; we only summarize a small collection below.

Observations.

(1) The algorithm detects trends at all levels of the tree, depending on the volume constraint given. When the volume constraint is large, naturally it produces trends at higher levels of the tree, and when the volume constraint is small, it produces (many more) trends at lower levels of the tree. Even for a fixed volume constraint, the nodes produced occur at various levels in the tree. This is an important feature, since given a volume constraint, the algorithm is able to pick nodes at all levels to best explain the most important trends. Note that it is trivial to simply ask for bursts at the root node or near the leaf levels.

(2) All the trends reported can be easily seen to correspond to the highlight news stories from the financial markets during the last ten years.

(3) As can be seen from Tables 2 and 3, working either with NATIVE-SP428-TREE or with BARTAL-SP428-TREE leads to very similar results.

[1]In stock trading terminology, the word "volume" has an entirely different meaning, referring to the number of shares traded in each day. This clash of terminology is unfortunate.

Node	Start Date	Stop Date	Volume
Energy	23-Aug-00	29-Nov-00	6443
Pharmaceuticals	25-Oct-00	4-Jan-01	7097
Financials	5-Dec-00	21-Feb-01	14476
Information Technology	2-Jun-00	25-Sep-00	23940
S&P	18-Apr-01	30-Aug-01	117799
Industrials	18-Dec-01	22-Apr-02	11276
Financials	6-Mar-02	28-May-02	15843
Computers & Peripherals	4-Dec-01	18-Jan-02	2339
Industrials	25-Nov-03	20-Feb-04	8436
Consumer Discretionary	12-Aug-03	23-Feb-04	15443
Consumer Staples	7-Oct-03	19-Feb-04	17199
Health Care	13-Nov-03	20-Feb-04	12712
Health Care	4-Jun-03	1-Aug-03	7762
Financials	16-Sep-03	23-Feb-04	31777
Information Technology	18-Dec-03	19-Feb-04	7393
S&P	17-Oct-02	21-Jan-03	69608
S&P	8-Aug-02	12-Sep-02	26741
S&P	23-Jun-98	3-Aug-98	28143

Table 1: Trend analysis using NATIVE-SP428-TREE. $v_{min} = 20000$

Nodes	Start date	Stop date	Volume
Hardware & Software	17-Aug-00	11-Sep-00	5364
Hardware & Software	18-Jan-01	8-Feb-01	4153
Capital Goods	3-Apr-00	8-Jun-00	8687
S&P 500	18-Apr-01	30-Aug-01	117799
Technology	13-Nov-01	26-Apr-02	127102
Technology	6-Oct-03	20-Feb-04	109154
Energy	12-Aug-03	23-Feb-04	15762
S&P 500	17-Oct-02	21-Jan-03	69608
S&P 500	8-Aug-02	12-Sep-02	26741
S&P 500	23-Jun-98	3-Aug-98	28143

Table 2: Trend analysis using BARTAL-SP428-TREE, $v_{min} = 20000$

(3) The upswings in the technology and computers stocks during 1999–2001, as well as the downtrend in the technology sector and the corresponding upward trend in other sectors like capital goods, health care, etc., are identified.

(5) In each case, when the volume constraint is set low enough, the algorithm identifies the prototypical examples of sub-industries or individual companies that contributed significantly to the trend.

Synthetic basket data.

We illustrate the performance on the synthetic basket data in Table 3. Here, the parameters $\delta = 10^{-5}$, $j_1 = 1.5 \times 10^{-5}, j_2 = 2.5 \times 10^{-5}, j_3 = 3.5 \times 10^{-5}$, $f_1 = 0.03, f_2 = 0.04, f_3 = 0.05$. The overall "sales" volume is plotted in Figure 1.

As can be seen from Table 3, several trends are identified at various combinations of internal nodes of the three trees, and furthermore, the trends identified naturally correspond to bursts of activity in the overall volume plot. In the trends identified, the volume, as well as the sizes of the sub-trees in each of the dimension, are seen to be varying at different time intervals. Finally, the algorithm was seen to scale well to the case of even larger data, which we do not include here.

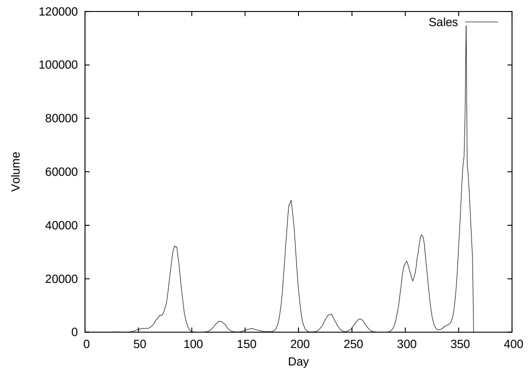

Figure 1: Volume plot for basket data

Node	Start day	Stop day	Volume
(95 222 311)	357	358	126047
(95 220 314)	357	358	6120
(95 214 315)	356	358	14358
(95 224 300)	191	196	198984
(95 208 314)	191	195	33890
(95 222 311)	83	88	148485
(91 222 313)	228	233	32720
(95 224 306)	125	130	21601
(93 222 315)	256	261	25006
(95 222 315)	83	88	148485

Table 3: Trend analysis using basket data. $v_{min} = 30000$

6. CONCLUSIONS

Our work offers the first, modular development of a mathematical and algorithmic framework for trend analysis problems. By a careful combination of various algorithmic paradigms (tree metric approximation, data stream models with sorting, Markov chain analysis), we have proposed a solution for trend analysis in massive data sets; our algorithms do not need large-scale reorganization of data, and only rely on auxiliary information that could be pre-computed with some domain expertise. We have also managed to achieve a high degree of abstraction, which makes our framework and algorithm suitable in a variety of applications. Our basket data generator based on the random walk model is of independent interest, and might also serve as a benchmark for future studies on temporal data mining, especially for trend discovery and analysis.

7. REFERENCES

[1] C. Aggarwal. A framework for diagnosing changes in evolving data streams. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 575–586, 2003.

[2] G. Aggarwal, M. Datar, S. Rajagopalan, and M. Ruhl. On models for massive data set computations, 2003. Manuscript.

[3] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the 1998 ACM SIGMOD International*

Conference on Management of Data, pages 94–105, 1998.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 1994 Internationall Conference on Very Large Data Bases*, pages 487–499, 1994.

[5] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

[6] Y. Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 184–193, 1996.

[7] Y. Bartal. On approximating arbitrary metrices by tree metrics. In *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing*, pages 161–168, 1998.

[8] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80, 2000.

[9] P. Domingos and G. Hulten. Catching up with the data: Research issues in mining data streams. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2001.

[10] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate L^1-difference algorithm for massive data streams. *SIAM Journal on Computing*, 32:131–151, 2002.

[11] V. Ganti, J. Gehrke, and R. Ramakrishnan. Mining data streams under block evolution. *ACM SIGKDD Explorations*, 3(2):1–10, 2002.

[12] A. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 389–398, 2002.

[13] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB)*, pages 79–88, 2001.

[14] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, 2001.

[15] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 97–106, 2001.

[16] J. Kleinberg. Bursty and hierarchical structure in streams. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 91–101, 2002.

[17] V.S. Lakshmanan, R.T. Ng, C.X. Wang, X. Zhou, and T.J. Johnson. The generalized MDL approach for summarization. In *Proceedings of the 2002 Internationall Conference on Very Large Data Bases*, 2002.

[18] Standard & Poor's. See `http://www.standardandpoors.com`.

[19] J. Yang and J. Widom. Incremental computation and maintenance of temporal aggregates. In *Proceedings of the 17th International Conference on Data Engineering*, pages 51–60, 2001.

APPENDIX

A. BARTAL'S METRIC APPROXIMATIONS

The following definitions and theorems are paraphrased from [6, 7].

Let V be a finite set with n points, and let M be a metric on V. A metric space N over V is said to *dominate* M if, for every $u, v \in V$, $d_N(u,v) \geq d_M(u,v)$; N is said to *α-approximate* M if N dominates M and for every $u, v \in V$, $d_N(u,v) \leq \alpha d_M(u,v)$. A family of metric spaces $\mathcal{N}$ over V is said to *α-probabilistically approximate* M if every metric in $\mathcal{N}$ dominates M and there is some probability distribution over $N \in \mathcal{N}$ such that for every $u, v \in V$, $\mathrm{E}[d_N(u,v)] \leq \alpha d_M(u,v)$.

A *tree metric* U over V is a metric space presented as a edge-weighted tree U_V whose leaves are the points of V, and where for any $u, v \in V$, $d_U(u,v)$ is defined to be the sum of the shortest path from u to v in the tree U_V. A *k-hierarchically well separated tree* is defined as a rooted weighted tree with the following properties: (1) the edge weight from any node to each of its children is the same, and (2) the edge weights along any path from the root to a leaf are decreasing by a factor of at least k.

THEOREM 2 ([6, 7]). *Any finite metric V on n points can be α-probabilistically approximated by a set of k-HSTs where $\alpha = O(k \log n \log\log n)$. Each HST in the family has diameter proportional to G, and moreover, the construction of the HST and the probability distribution on the family of HSTs can be done in time polynomial in n.*