Proc. VLDB 02

Multi-Dimensional Regression Analysis of Time-Series Data Streams*

Yixin Chen[1] **Guozhu Dong**[2] **Jiawei Han**[1] **Benjamin W. Wah**[1] **Jianyong Wang**[1]

[1] University of Illinois at Urbana-Champaign, U.S.A.
[2] Wright State University, U.S.A.

Abstract

Real-time production systems and other dynamic environments often generate tremendous (potentially infinite) amount of stream data; the volume of data is too huge to be stored on disks or scanned multiple times. Can we perform on-line, multi-dimensional analysis and data mining of such data to alert people about dramatic changes of situations and to initiate timely, high-quality responses? This is a challenging task.

In this paper, we investigate methods for on-line, multi-dimensional regression analysis of time-series stream data, with the following contributions: (1) our analysis shows that only a small number of compressed regression measures instead of the complete stream of data need to be registered for multi-dimensional linear regression analysis, (2) to facilitate on-line stream data analysis, a partially materialized data cube model, with *regression* as measure, and a *tilt time frame* as its time dimension, is proposed to minimize the amount of data to be retained in memory or stored on disks, and (3) an exception-guided drilling approach is developed for on-line, multi-dimensional exception-based regression analysis. Based on this design, algorithms are proposed for efficient analysis of time-series data streams. Our performance study compares the proposed algorithms and identifies the most memory- and time- efficient one for multi-dimensional stream data analysis.

* The work was supported in part by grants from U.S. National Science Foundation, the University of Illinois, and Microsoft Research.

Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002

1 Introduction

With years of research and development of data warehouse and OLAP technology [12, 7], a large number of data warehouses and data cubes have been successfully constructed and deployed in applications, and data cube has become an essential component in most data warehouse systems and in some extended relational database systems and has been playing an increasingly important role in data analysis and intelligent decision support.

The data warehouse and OLAP technology is based on the integration and consolidation of data in multi-dimensional space to facilitate powerful and fast on-line data analysis. Data are aggregated either completely or partially in multiple dimensions and multiple levels and are stored in the form of either relations or multi-dimensional arrays [1, 28]. The dimensions in a data cube are of categorical data, such as products, region, time, etc., and the measures are numerical data, representing various kinds of aggregates, such as *sum*, *average*, and *variance* of sales or profits, etc.

The success of OLAP technology naturally leads to its possible extension from the analysis of static, pre-integrated, historical data to that of current, dynamically changing data, including time-series data, scientific and engineering data, and data produced in other dynamic environments, such as power supply, network traffic, stock exchange, tele-communication data flow, Web click streams, weather or environment monitoring, etc. However, a fundamental difference in the analysis in a dynamic environment from that in a static one is that the dynamic one relies heavily on regression and trend analysis instead of simple, static aggregates. The current data cube technology is good for computing static, summarizing aggregates, but is not designed for regression and trend analysis. "*Can we extend the data cube technology and construct some special kind of data cubes so that regression and trend analysis can be performed efficiently in the multi-dimensional space?*" This is the task of this study.

In this paper, we examine a special kind of dynamic data, called *stream data*, with time-series as its representative. Stream data is generated continuously in a dynamic environment, with huge volume, infinite flow, and fast changing behavior. As collected, such data is almost always at rather low level, consisting of various kinds of detailed temporal and other features. To find interesting or unusual patterns, it is essential to perform regression analysis at certain meaningful abstraction level, discover critical changes of data, and drill down to some more detailed levels for in-depth analysis, when needed.

Let's examine an example.

Example 1 A power supply station *"collects"* infinite streams of power usage data, with the lowest granularity as (individual) user, location, and minute. Given a large number of users, it is only realistic to analyze the fluctuation of power usage at certain high levels, such as by city or district and by hour, making timely power supply adjustments and handling unusual situations.

Conceptually, for multi-dimensional analysis, one can view such stream data as a *virtual* data cube, consisting of one measure[1], *regression*, and a set of dimensions, including one *time dimension*, and a few *"standard" dimensions*, such as location, user-category, etc. However, in practice, it is impossible to materialize such a data cube, since the materialization requires a huge amount of data to be computed and stored. Some efficient methods must be developed for systematic analysis of such data. ■

In this study, we take Example 1 as a typical scenario and study how to perform efficient and effective multi-dimensional regression analysis of stream data, with the following contributions.

1. Our study shows that for linear and multiple linear regression analysis, only a small number of *regression measures* rather than the complete stream of data need to be used. This holds for regression on both the time dimension and the other (standard) dimensions. Since it takes a much smaller amount of space and time to handle regression measures in a multi-dimensional space than handling the stream data itself, it is preferable to construct regression(-measured) cubes by computing such regression measures.

2. For on-line stream data analysis, both space and time are critical. In order to avoid imposing unrealistic demand on space and time, instead of computing a fully materialized regression cube, we suggest to compute a partially materialized data cube, with *regression* as measure, and a *tilt time frame* as its time dimension. In the *tilt time frame*, time is registered at different levels of granularity. The most recent time is registered at the finest granularity; the more distant time is registered at coarser granularity; the level of coarseness depends on the application requirements and on how old the time point is. This model is sufficient for most analysis tasks, and at the same time it also ensures that the total amount of data to retain in memory is small.

3. Due to limited memory space in stream data analysis, it is often too costly to store a precomputed regression cube, even with the *tilt time frame*. We propose to compute and store only two critical layers (which are essentially cuboids) in the cube: (1) an observation layer, called *o-layer*, which is the layer that an analyst (or the system) checks and makes decisions for either signaling the exceptions, or drilling on the exception cells down to lower layers to find their corresponding exception "supporters"; and (2) the minimal interesting layer, called *m-layer*, which is the minimal layer that an analyst would like to study, since it is often neither cost-effective nor practically interesting to examine the minute detail of stream data. For example, in Ex. 1, we assume the *o*-layer is *city* and *hour*, while the m-layer is *street-block* and *quarter* (of hour).

4. Storing a regression cube at only two critical layers leaves a lot of room for approaches for computing the cuboids between the two layers. We propose two alternative methods for handling the cuboids in between: (1) computing cuboids from the m-layer to the *o*-layer but retaining only the computed exception cells in between; we call this method the m/o-cubing method; and (2) rolling-up the cuboids from the m-layer to the *o*-layer, by following one popular drilling path, and computing other exception cells using the the computed cuboids along the path; we call this method the popular-path cubing method. Our performance study shows that both methods require a reasonable amount of memory and have quick aggregation time and exception detection time. Our analysis also compares the strength and weakness of the two methods.

The rest of the paper is organized as follows. In Section 2, we define the basic concepts and introduce the research problem. In Section 3, we present the theoretic foundation for computing multiple linear regression models in data cubes. In Section 4, the concepts of *tilt time frame* and *critical layers* are introduced for regression analysis of stream data, and two cuboid computation methods, m/o-cubing and popular-path cubing, are presented with their strength and weakness analyzed and compared. Our experiments and performance study of the methods are presented in Section 5. The related work and possible extensions of the model are discussed in Section 6, and our study is concluded in Section 7.

[1] A cube may contain other measures, such as total power usage. Since such measures have been analyzed thoroughly, we will not include them in our discussion here.

2 Problem Definition

In this section, we introduce the basic concepts related to linear regression analysis in time-series data cubes and define our problem for research.

2.1 Data cubes

Let $\mathcal{D}$ be a relational table, called the base table, of a given cube. The set of all attributes $\mathcal{A}$ in $\mathcal{D}$ are partitioned into two subsets, the dimensional attributes DIM and the measure attributes M (so $DIM \cup M = \mathcal{A}$ and $DIM \cap M = \emptyset$). The measure attributes functionally depend on the dimensional attributes in $\mathcal{D}$ and are defined in the context of data cube using some typical aggregate functions, such as COUNT, SUM, AVG, or some regression related measures to be studied here.

A tuple with schema $\mathcal{A}$ in a multi-dimensional space (i.e., in the context of data cube) is called a **cell**. Given three distinct cells c_1, c_2 and c_3, c_1 is an **ancestor** of c_2, and c_2 a **descendant** of c_1 iff on every dimensional attribute, either c_1 and c_2 share the same value, or c_1's value is a generalized value of c_2's in the dimension's concept hierarchy. c_2 is a **sibling** of c_3 iff c_2 and c_3 have identical values in all dimensions except one dimension A where $c_1[A]$ and $c_2[A]$ have the same parent in the dimension's domain hierarchy. A cell which has k non-* values is called a **k-d cell**. (We use "$*$" to indicate "all", i.e., the highest level on any dimension.)

A tuple $c \in \mathcal{D}$ is called a **base cell**. A base cell does not have any descendant. A cell c is an **aggregated cell** iff it is an ancestor of some base cell. For each aggregated cell c, its values on the measure attributes are derived from the complete set of descendant base cells of c.

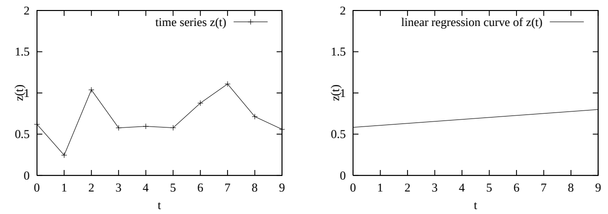
2.2 Time series in data cubes

A time series is a sequence (or function) $z(t)$ that maps each time point to some numerical value[2]; each time series has an associated time interval $z(t) : t \in [t_b, t_e]$, where t_b is the starting time and t_e is the ending time. We only consider discrete time, and so $[t_b, t_e]$ here represents the sequence of integers starting from t_b and ending at t_e.

Example 2 The sequence $z(t)$: 0.62, 0.24, 1.03, 0.57, 0.59, 0.57, 0.87, 1.10, 0.71, 0.56 is a time series over any time interval of 10 time points, e.g. $[0, 9]$. Figure 1 (a) is a diagram for this time series. ■

In time series analysis, a user is usually interested in finding dominant trends or comparing time series to find similar or dissimilar curves. A basic technique popularly used for such analyses is linear regression.

Figure 1 (b) shows the linear regression of the time series given in Figure 1 (a), which captures the main trend of the time series.



a) $z(t)$ b) Linear regression of $z(t)$

Figure 1: A time series $z(t)$ and its linear regression

Previous research considered how to compute the linear regression of a time series. However, to the best of our knowledge, there is no prior work that considers linear regression of time series in a structured environment, such as a data cube, where there are a huge number of inter-related cells, which may form a huge number of analyzable time-series.

2.3 Multi-dimensional analysis of stream data

In this study, we consider stream data as huge volume, infinite flow of time-series data. The data is collected at the most detailed level in a multi-dimensional space. Thus the direct regression of data at the most detailed level may generate a large number of regression lines, but still cannot tell the general trends contained in the data.

Our task is to perform high-level, on-line, multi-dimensional analysis of such data streams in order to find unusual (exceptional) changes of trends, according to users' interest, and based on multi-dimensional linear regression analysis.

3 Foundations for Computing Linear Regression in Data Cubes

After reviewing the basics of linear regression, we introduce a compact representation of time series data for linear regression analysis in a data cube environment, and establish the aggregation formulae for computing linear regression models of time series of all cells using the selected materialized cells.[3] This enables us to obtain a compact representation of any aggregated cell from that of given descendant cells, and provides a theoretic foundation for warehousing linear regression models of time series.

For the sake of simplicity, we only discuss linear regression of time series here. In the full paper, we consider the general case of multiple linear regression for general stream data with more than one regression variable and/or with irregular time ticks.

[2] Time series can be more involved. In this paper we restrict ourselves to this simple type of time series.

[3] Only proof sketches are provided; detailed proofs are included in the full paper.

3.1 Linear regression for one time series

Here we briefly review the fundamentals of linear regression for the case involving just one time series.

A linear fit for a time series $z(t) : t \in [t_b, t_e]$ is a linear estimation function:

$$\hat{z}(t) = \hat{\theta} + \hat{\eta} t$$

where $\hat{z}(t)$ is the estimated value of $z(t)$, and $\hat{\theta}$ (the base) and $\hat{\eta}$ (the slope) are two parameters. The difference $z(t) - \hat{z}(t)$ is the residual for time t.

Definition 1 *The* least square error (LSE) linear fit *of a time series $z(t)$ is a linear fit where $\hat{\theta}$ and $\hat{\eta}$ are chosen to minimize the* residual sum of squares*:* $RSS(\hat{\theta}, \hat{\eta}) = \sum_{t=t_b}^{t_e} [z(t) - (\hat{\theta} + \hat{\eta} t)]^2$.

Lemma 3.1 *The parameters for the LSE linear fit can be obtained as follows:*

$$\hat{\eta} = \sum_{t=t_b}^{t_e} \left(\frac{t - \bar{t}}{SVS} \right) (z(t) - \bar{z}) = \sum_{t=t_b}^{t_e} \left(\frac{t - \bar{t}}{SVS} \right) z(t) \quad (1)$$

$$\hat{\theta} = \bar{z} - \hat{\eta} \bar{t} \quad (2)$$

where (SVS denotes the sum of variance squares *of t):* $SVS = \sum_{t=t_b}^{t_e} (t - \bar{t})^2 = \sum_{t=t_b}^{t_e} (t - \bar{t}) t$, $\bar{z} = \frac{\sum_{t=t_b}^{t_e} z(t)}{t_e - t_b + 1}$, $\bar{t} = \frac{\sum_{t=t_b}^{t_e} t}{t_e - t_b + 1} = \frac{t_b + t_e}{2}$.

3.2 Compact representations

As far as linear regression analysis is concerned, the time series of a data cube cell can be represented by either of the following two compressed representations:

- The *ISB representation* of the time series consists of $([t_b, t_e], \hat{\eta}, \hat{\theta})$, where $[t_b, t_e]$ is the **i**nterval for the time series, and $\hat{\eta}$ and $\hat{\theta}$ are the **s**lope and **b**ase of the linear fit for the time series.

- The *IntVal representation* of the time series consists of $([t_b, t_e], z_b, z_e)$, where $[t_b, t_e]$ is the **int**erval for the time series, and z_b and z_e are the **val**ues of the linear fit at time t_b and t_e, respectively.

These two representations are equivalent in the sense that one can be derived from the other. Thus only the ISB representation is used here. Moreover, as we will show later, by storing the ISB representation of the base cells of the cube, we can compute the linear regression models of all cells. This enables us to aggregate the data cuboids without retrieving the original time series data, and without any loss of precision.

Theorem 3.1 *(a) By materializing the ISB representation of the base cells of a cube, we can compute the ISB representation of all cells in the cube. (b) Moreover, this materialization is minimal—by materializing any proper subset of the ISB representation, one cannot obtain the linear regression models of all cells.*

Proof. We will prove (a) in sections 3.3 and 3.4.

For (b), it suffices to show that, one cannot even obtain the linear regressions of all base cells, using any proper subset of the ISB representation of the linear regressions of the base cells. For this it suffices to show that the components of the ISB representation are independent of each other. That is, for each proper subset of the ISB representation, there are two linear regressions, realized by two time series z_1 and z_2, which have identical values on this subset but have different values on the other components of the ISB representation. To show that t_b cannot be excluded, consider the time series $z_1 : 0, 0, 0$ over $[0, 2]$ and $z_2 : 0, 0$ over $[1, 2]$; their linear regressions agree on $t_e, \hat{\theta}, \hat{\eta}$ but not on t_b. Similarly, t_e cannot be removed. For $\hat{\theta}$, consider $z_1 : 0, 0$ and $z_2 : 1, 1$ over $[0, 1]$; $t_b = 0, t_e = 1, \hat{\eta} = 0$ for both, but $\hat{\theta} = 0$ for z_1 and $\hat{\theta} = 1$ for z_2. For $\hat{\eta}$, consider $z_1 : 0, 0$ and $z_2 : 0, 1$ over $[0, 1]$; $t_b = 0, t_e = 1, \hat{\theta} = 0$ for both, but $\hat{\eta} = 0$ for z_1 and $\hat{\eta} = 1$ for z_2. ■

The theorem does not imply that the ISB representation is the minimum necessary. The theoretical problem of whether there is a more compact representation (using fewer than 4 numbers) than ISB is open.

3.3 Aggregation on standard dimensions

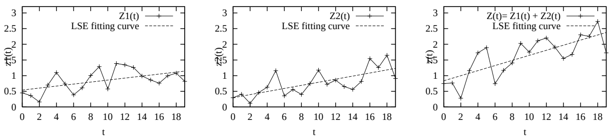
In this section and the next we consider how to derive the ISB representation of aggregated cells in a data cube, from the ISB representations of the relevant descendant (base or otherwise) cells[4]. Here we consider the case when the aggregated cells are obtained using aggregation (roll-up) on a standard dimension.

Let c_a be a cell aggregated (on a standard dimension) from a number of descendant cells $c_1, \ldots, c_K$. Here, the time series for c_a is defined to be the summation of the time series for the given descendant cells. That is, $z(t) = \sum_{i=1}^{K} z_i(t)$, where $z(t) : t \in [t_b, t_e]$ denotes the time series for c_a, and $z_i(t) : t \in [t_b, t_e]$ denotes that for c_i. Figure 2 gives an example.

The following theorem shows how to derive the ISB representation of the linear regression for c_a using the ISB representations of the descendant cells. In the theorem, $([t_b^a, t_e^a], \hat{\theta}_a, \hat{\eta}_a)$ denotes the ISB representation of c_a, and $([t_b^1, t_e^1], \hat{\theta}_1, \hat{\eta}_1)$, ..., $([t_b^K, t_e^K], \hat{\theta}_K, \hat{\eta}_K)$ denote the ISB representations of $c_1, ..., c_K$, respectively.

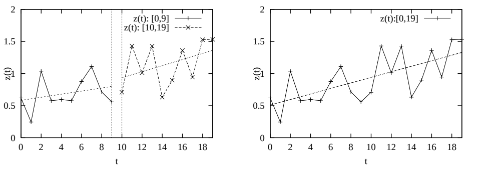
Theorem 3.2 Aggregations on standard dimensions. *For aggregations on a standard dimension, the ISB representation of the aggregated cell can be derived*

[4]When the descendant cells are not base cells, they should have disjoint sets of descendant base cells to ensure the correctness of aggregation.



a) $z_1(t)$ b) $z_2(t)$ c) $z(t) = z_1(t) + z_2(t)$

Figure 2: Example of aggregation on a standard dimension of $K = 2$ descendant cells. The ISB representation is ([0,19],0.540995, 0.0318379) for $z_1(t)$, ([0,19], 0.294875, 0.0493375) for $z_2(t)$, and ([0,19], 0.83587, 0.0811754) for $z(t)$. These satisfy Theorem 3.2.



a) two time intervals b) aggregated interval

Figure 3: Example of aggregation on the time dimension of two time intervals. The ISB representations are ([0,9], 0.582995, 0.0240189), ([10,19], 0.459046, 0.047474), and ([0,19], 0.509033, 0.0431806). These satisfy Theorem 3.3.

from that of the descendant cells as follows: (a) $[t_b^a, t_e^a] = [t_b^1, t_e^1]$, *(b)* $\hat{\eta}_a = \sum_{i=1}^K \hat{\eta}_i$, *(c)* $\hat{\theta}_a = \sum_{i=1}^K \hat{\theta}_i$.

Proof. Statement (a) is obvious.

Let $n_a = t_e - t_b + 1$, $\bar{z}_a = \frac{\sum_{t=t_b}^{t_e} z(t)}{n_a}$, $\bar{z}_i = \frac{\sum_{t=t_b}^{t_e} z_i(t)}{n_a}$ (for $i = 1...K$), and $\bar{t} = \frac{\sum_{t=t_b}^{t_e} t}{n_a}$.

Statement (b) holds because

$$\begin{aligned}\hat{\eta}_a &= \sum_{t=t_b}^{t_e} \left(\frac{t-\bar{t}}{\sum_{t=t_b}^{t_e}(t-\bar{t})^2}(z(t) - \bar{z}_a) \right) \\ &= \sum_{t=t_b}^{t_e} \left(\frac{t-\bar{t}}{\sum_{t=t_b}^{t_e}(t-\bar{t})^2}\left(\sum_{i=1}^K z_i(t) - \sum_{i=1}^K \bar{z}_i\right) \right) \\ &= \sum_{i=1}^K \left[\sum_{t=t_b}^{t_e} \left(\frac{t-\bar{t}}{\sum_{t=t_b}^{t_e}(t-\bar{t})^2}(z_i(t) - \bar{z}_i) \right) \right] \\ &= \sum_{i=1}^K \hat{\eta}_i\end{aligned}$$

Statement (c) holds because

$$\begin{aligned}\hat{\theta}_a &= \bar{z}_a - \hat{\eta}_a\bar{t} = \sum_{i=1}^K \bar{z}_i - \sum_{i=1}^K \hat{\eta}_i\bar{t} \\ &= \sum_{i=1}^K \left(\bar{z}_i - \hat{\eta}_i\bar{t} \right) = \sum_{i=1}^K \hat{\theta}_i \quad \blacksquare\end{aligned}$$

The regressions in Figure 2 illustrate this theorem.

3.4 Aggregation on the time dimension

In this section we consider how to derive the ISB representation of aggregated cells in a data cube, from the ISB representations of the relevant descendant cells, for the case when the aggregated cells are obtained using aggregation (roll-up) on the time dimension.

Let c_a be the aggregated cell, and $c_1, ..., c_K$ the descendant cells. These cells should be related as follows: (1) The time intervals $[t_b^1, t_e^1]$, ..., $[t_b^K, t_e^K]$ of $c_1, \cdots, c_K$ should form a partition of the time interval $[t_b, t_e]$ of c_a. (2) Let $z(t) : t \in [t_b, t_e]$ denote the time series of c_a. Then $z(t) : [t_b^i, t_e^i]$ is time series of c_i. Without loss of generality, we assume that $t_b^i < t_b^{i+1}$ for all $1 \leq i < K$. Figure 3 gives an example, where $[t_b, t_e] = [0, 19]$, $[t_b^1, t_e^1] = [0, 9]$, and $[t_b^2, t_e^2] = [10, 19]$.

Similar to notations used in section 3.3, we write $([t_b^a, t_e^a], \hat{\theta}_a, \hat{\eta}_a)$ for the ISB representation of c_a, $([t_b^i, t_e^i], \hat{\theta}_i, \hat{\eta}_i)$ for that of c_i. Moreover, we introduce 8 variables: let $n_a = t_e - t_b + 1$, $n_i = t_e^i - t_b^i + 1$, $S_a = \sum_{t=t_b}^{t_e} z(t)$, $S_i = \sum_{t=t_b^i}^{t_e^i} z(t)$, $\bar{z}_a$ denote the average of $z(t) : t \in [t_b, t_e]$, $\bar{z}_i$ that of $z(t) : t \in [t_b^i, t_e^i]$, $\bar{t}_a$ that of $t \in [t_b, t_e]$, and $\bar{t}_i$ that of $t \in [t_b^i, t_e^i]$.

All of these 8 variables can be computed from the ISBs of the c_i's. Indeed, $\bar{t}_i = \frac{1}{2} * (t_b^i + t_e^i)$, $\bar{t}_a = \frac{1}{2} * (t_b^1 + t_e^K)$, $\bar{z}_i = \hat{\theta}_i + \hat{\eta}_i\bar{t}_i$ (by Equation 2), $S_i = n_i * \bar{z}_i$, $S_a = \sum_{i=1}^K S_i$, $\bar{z}_a = \frac{S_a}{n_a}$. So we can use these variables for expressing the ISB of c_a.

Theorem 3.3 [Aggregations on the time dimension.] *For aggregations on the time dimension, the ISB representation of the aggregated cell can be derived from that of the descendant cells as follows:*

(a) $[t_b, t_e] = [t_b^1, t_e^K]$

(b) $\hat{\eta}_a = \sum_{i=1}^K \left(\frac{n_i^3 - n_i}{n_a^3 - n_a}\hat{\eta}_i \right)$

$$+6\sum_{i=1}^K \left(\frac{2\sum_{j=1}^{i-1} n_j + n_i - n_a}{n_a^3 - n_a} \frac{n_a S_i - n_i S_a}{n_a} \right)$$

(c) $\hat{\theta}_a = \bar{z}_a - \hat{\eta}_a\bar{t}_a$, *where* $\bar{z}_a$ *and* $\bar{t}_a$ *are derived as discussed above,* $\hat{\eta}_a$ *is derived as in (b).*

To prove the theorem, we will need a lemma.

Lemma 3.2 *For all integers* $i \geq 0$ *and* $n > 0$, *let* $\bar{j} = \sum_{j=i}^{i+n-1} j/n$. *Then* $\sum_{j=i}^{i+n-1}(j - \bar{j})^2 = \frac{n^3 - n}{12}$.

We will use $\alpha(n)$ to denote $\sum_{j=i}^{i+n-1}(j - \bar{j})^2$, which is independent of i by the lemma.

Proof (Sketch) (of Thm 3.3). (a) This is obvious.

(b) From Lemmas 3.1, we have

$$\begin{aligned}\hat{\eta}_a &= \sum_{t=t_b}^{t_e}\left[\frac{t-\bar{t}}{\alpha(n_a)}(z(t) - \bar{z})\right] \\ &= \sum_{i=1}^K \sum_{t=t_b^i}^{t_e^i}\left[\frac{t-\bar{t}}{\alpha(n_a)}(z(t) - \bar{z})\right].\end{aligned}$$

For each i, we have:

$$\begin{aligned}&\sum_{t=t_b^i}^{t_e^i}\left[\frac{t-\bar{t}}{\alpha(n_a)}(z(t) - \bar{z})\right] \\ &= \sum_{t=t_b^i}^{t_e^i}\left[\frac{t-\bar{t}_i}{\alpha(n_a)}(z(t) - \bar{z})\right] + \sum_{t=t_b^i}^{t_e^i}\left[\frac{\bar{t}_i-\bar{t}}{\alpha(n_a)}(z(t) - \bar{z})\right]\end{aligned}$$

We can prove (by including using Lemma 3.2) that

$$\sum_{t=t_b^i}^{t_e^i}\left[\frac{t-\bar{t}_i}{\alpha(n_a)}(z(t) - \bar{z})\right] = \frac{n_i^3 - n_i}{n_a^3 - n_a}\hat{\eta}_i,$$

and

$$\begin{aligned}&\sum_{t=t_b^i}^{t_e^i}\left[\frac{\bar{t}_i-\bar{t}}{\alpha(n_a)}(z(t) - \bar{z})\right] \\ &= 6 * \frac{2\sum_{j=1}^{i-1} n_j + n_i - n_a}{n_a^3 - n_a}\frac{n_a S_i - n_i S_a}{n_a}.\end{aligned}$$

So (b) is proven.

(c) By (b) and the discussion above this theorem, we know that all variables in the right-hand-side of $\hat{\theta}_a = \bar{z}_a - \hat{\eta}_a \bar{t}_a$, can be expressed in terms of the ISBs of the c_i's. ■

4 Stream Data Analysis with Regression Cubes

Although representing stream data by regression parameters (points) may substantially reduce the data to be stored and/or aggregated, it is still often too costly in both space and time to fully compute and materialize the regression points at a multi-dimensional space due to the limitations on resources and response time in on-line stream data analysis. In the following, we propose three ways to further reduce the cost: (1) *tilt time frame*, (2) notion of *critical layers*: o-layer (observation layer) and m-layer (minimal interesting layer), and (3) *exception-based computation and drilling*. We present two algorithms for performing such computation.

4.1 Tilt time frame

In stream data analysis, people are often interested in recent changes at a fine scale, but long term changes at a coarse scale. Naturally, one can register time at different levels of granularity. The most recent time is registered at the finest granularity; the more distant time is registered at coarser granularity; and the level of coarseness depends on the application requirements.

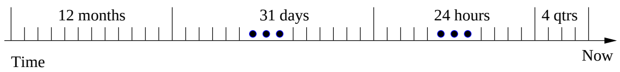


Figure 4: A tilt time frame model

Example 3 For Ex. 1, a *tilt time frame* can be constructed as shown in Figure 4, where the time frame is structured in multiple granularities: the most recent 4 quarters (15 minutes), then the last 24 hours, 31 days, and 12 months. Based on this model, one can compute regressions in the last hour with the precision of quarter of an hour, the last day with the precision of hour, and so on, until the whole year, with the precision of month[5]. This model registers only $4+24+31+12 = 71$ units of time instead of $366 \times 24 \times 4 = 35,136$ units, a saving of about 495 times, with an acceptable trade-off of the grain of granularity at a distant time. ■

4.2 Notion of critical layers

Even with the *tilt time frame* model, it could still be too costly to dynamically compute and store a full regression cube since such a cube may have quite a few standard dimensions, each containing multiple levels with many distinct values. Since stream data analysis has only limited memory space but requires fast response time, it is suggested to compute and store only some mission-critical cuboids in the cube.

[5] We align the time axis with the natural calendar time. Thus, for each granularity level of the tilt time frame, there might be a partial interval which is less than a full unit at that level.

In our design, two critical cuboids are identified due to their conceptual and computational importance in stream data analysis. We call these cuboids layers and suggest to compute and store them dynamically. The first layer, called *m-layer*, is the minimally interesting layer that an analyst would like to study. It is necessary to have such a layer since it is often neither cost-effective nor practically interesting to examine the minute detail of stream data. The second layer, called *o-layer*, is the observation layer at which an analyst (or an automated system) would like to check and make decisions of either signaling the exceptions, or drilling on the exception cells down to lower layers to find their lower-level exceptional descendants.

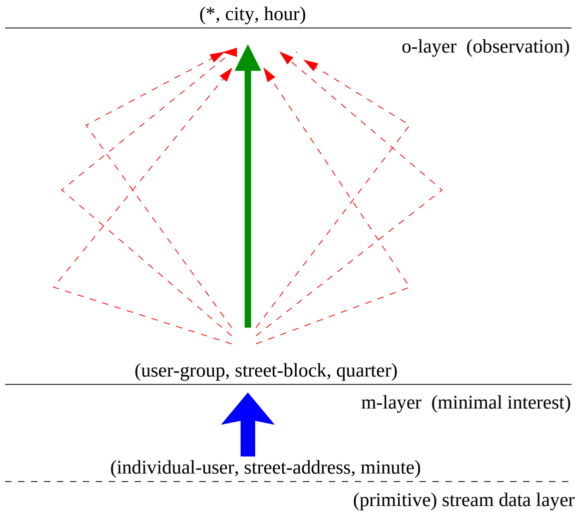


Figure 5: Two critical layers in the regression cube

Example 4 Assume that the (virtual) cuboid "(*individual_user*, *street_address*, *minute*)" forms the primitive layer of the input stream data in Ex. 1. With the *tilt time frame* as shown in Figure 4, the two critical layers for power supply analysis are: (1) the m-layer: (*user_group*, *street_block*, *quarter*), and (2) the o-layer: ($*$, *city*, *hour*), as shown in Figure 5.

Based on this design, the cuboids lower than the m-layer will not need to be computed since they are beyond the minimal interest of users. Thus the minimal regression cells that our base cuboid needs to be computed and stored will be the aggregate cells computed with grouping by *user_group*, *street_block*, and *quarter*. This can be done by aggregations of regressions (Section 3) (1) on two standard dimensions, *user* and *location*, by rolling up from *individual_user* to *user_group* and from *street_address* to *street_block*, respectively, and (2) on time dimension by rolling up from *minute* to *quarter*.

Similarly, the cuboids at the *o*-layer should be computed dynamically according to the tilt time frame model as well. This is the layer that an analyst takes as an observation deck, watching the changes of the current stream data by examining the regression lines and/or curves at this layer to make decisions. The layer can be obtained by rolling up the cube (1) along two standard dimensions to $*$ (which means *all* user_category) and *city*, respectively, and (2) along time dimension to *hour*. If something unusual is observed, the analyst can drill down the exceptional cells to examine low level details. ■

4.3 Framework for exception-based analysis

Materializing a regression cube at only two critical layers leaves a lot of room for approaches for computing the cuboids in between. These cuboids can be precomputed fully, partially, exception cells only, or not at all (leave everything to on-the-fly computation). Since there may be a large number of cuboids between these two layers and each may contain many cells, it is often too costly in both space and time to fully materialize these cuboids. Moreover, a user may be only interested in certain *exception cells* in those cuboids. Thus it is desirable to compute only such exception cells.

A regression line is exceptional if its slope is $\geq$ the exception threshold, where an exception threshold can be defined by a user or an expert for each cuboid c, for each dimension level d, or for the whole cube, depending on applications. Moreover, the regression line may refer to the regression represented by one cell itself in a cuboid, or between two points represented by the current cell (such as the current quarter) vs. the previous one (the last quarter), or the current hour vs. the last, etc. In general, regression is computed against certain points in the tilt time frame, based on users' interest and application.

According to the above discussion, we propose the following framework in our computation.

Framework 4.1 (Exception-driven analysis) The task of computing a regression-based time-series cube is to (1) compute two critical layers (cuboids): i) *m-layer* (the minimal interest layer), and ii) *o-layer* (the observation layer), and (2) for the cuboids between the two layers, compute only those *exception cells* (i.e., the cells which pass an exception threshold) that have at least one exception parent (cell). ■

This framework, though not covering the full search space, is a realistic one because with a huge search space in a cube, a user rarely has time to examine normal cells at the layer lower than the observation layer, and it is natural to follow only the exceptional cells to drill down and check only their exceptional descendants in order to find the cause of the problem(s). Based on this framework, one only needs to compute and store a small number of exception cells that satisfy the condition. Such cost reduction makes possible the OLAP-styled, regression-based exploration of cubes in stream data analysis.

4.4 Algorithms for exception-based regression cube computation

Based on the above discussion, we have the following algorithm design for efficient computation of exception-based regression cubes.

First, since the tilt time frame is used in time dimension, the regression data is computed based on the tilt time frame. The determination of *exception threshold* and *the reference points for the regression lines* should be dependent on the particular application.

Second, as discussed before, the m-layer should be the layer aggregated directly from the stream data.

Third, a compact data structure needs to be designed so that the space taken in the computation of aggregations is minimized. Here a data structure, called H-tree, a hyper-linked tree structure introduced in [18], is revised and adopted to ensure that a compact structure is maintained in memory for efficient computation of multi-dimensional and multi-level aggregations.

We present these ideas using an example.

Example 5 Suppose the stream data to be analyzed contains 3 dimensions, A, B and C, each with 3 levels of abstraction (excluding the highest level of abstraction "$*$"), as (A_1, A_2, A_3), (B_1, B_2, B_3), (C_1, C_2, C_3), where the ordering of $* > A_1 > A_2 > A_3$ forms a high-to-low hierarchy, and so on. The minimal interesting layer (the m-layer) is (A_2, B_2, C_2), and the o-layer is $(A_1, *, C_1)$. From the m-layer (the bottom cuboid) to the o-layer (the top-cuboid to be computed), there are in total $2 \times 3 \times 2 = 12$ cuboids, as shown in Figure 6.

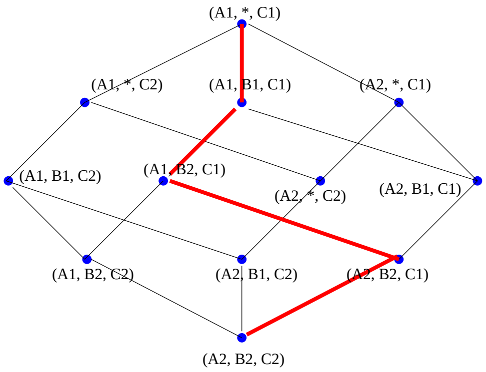


Figure 6: Cube structure from the m-layer to the o-layer

Assume that the cardinality (the number of distinct values) at each level has the relationship: $card(A_1) < card(B_1) < card(C_1) < card(C_2) < card(A_2) < card(B_2)$. Then each path of an H-tree from root to leaf is ordered as $\langle A_1, B_1, C_1, C_2, A_2, B_2\rangle$. This ordering makes the tree compact since there are likely more sharings at higher level nodes. Each tuple, expanded to include ancestor values of each dimension value, is inserted into the H-tree, as a path with nodes (namely attribute-value pairs) in this order. An example H-tree is shown in Fig 7. In the leaf node of each path (e.g. $\langle a_{11}, b_{11}, c_{11}, c_{21}, a_{21}, b_{21}\rangle$), we store relevant regression information (namely the ISBs) of the cells of the m-layer. The upper level regressions are computed using the H-tree and its associated links and header tables. A header table is constructed for (a subset of) each level of the tree, with entries containing appropriate statistics for cells (e.g. $(*, b_{21}, *)$ for the bottom level in Fig 7) and a linked list of all nodes contributing to the cells. ■

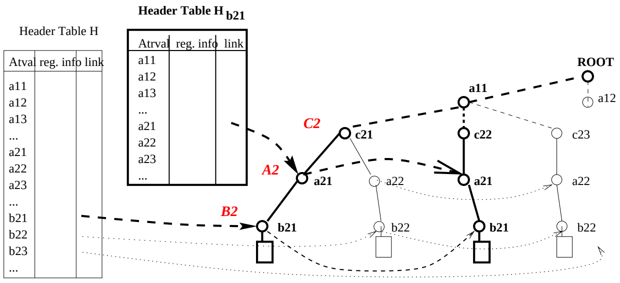


Figure 7: H-tree structure for cube computation

Now our focus becomes how to compute the o-layer and the exception cells between m- and o- layers. Two interesting methods are outlined as follows.

1. m/o-cubing: Starting at the m-layer, cubing is performed by aggregating regression cells upto the o-layer using an efficient cubing method, such as multiway array aggregation [28], BUC [5], or H-cubing [18]. In our implementation, H-cubing is performed using the H-tree constructed. Only the exception cells are retained after using the corresponding layer in computation (except for the o-layer in which all cells are retained for observation).

2. popular-path cubing: Starting at the m-layer, aggregation of regression cells upto the o-layer is performed by following a popular drilling path using an efficient cubing method, such as H-cubing [18]. Then for other cuboids between m and o-layers, only the children cells of an exception cell of a computed cuboid need to be computed, with the newly computed exception cells retained. Such computation may utilize the precomputed cuboids along the popular path.

Here we present the two algorithms.

Algorithm 1 (m/o H-cubing) H-cubing for computing regressions from the m- to the o- layers.

Input. Multi-dimensional time-series stream data, plus (1) the m and o-layer specifications, and (2) exception threshold(s) (possibly one per level or cuboid).

Output. All the regression cells at the m/o-layers and the exception cells for the layers in between.

Method.

1. Aggregate stream data to the m-layer, based on Theorems 3.2 and 3.3, and construct H-tree by scanning stream data once and performing aggregation in the corresponding leaf nodes of the tree.

2. Compute aggregation starting at the m-layer and ending at the o-layer, using the H-cubing method[6] described in [18]. The leaf-level header table (corresponding to 1-d cells) is used for building the header table for the next level (corresponding to 2-d cells), and so on. When a local H-header computation is finished, output only the exception cells.
   Notice that H-cubing proceeds from the leaf nodes up, level by level, computing each combination of distinct values at a particular dimension-level combination, by traversing the node-links and performing aggregation on regression.
   For example, when traversing following the links of node b_{21}, the local header table $H_{b_{21}}$ is used to hold the aggregated value for (b_{21}, a_{21}), (b_{21}, a_{22}), etc. However, when traversing following the links of b_{22}, the same local header table space is reused as header $H_{b_{22}}$ and thus the space usage is minimized. Only the exception cells in the (local) header tables for each combination will be output.

Analysis. Step 1 needs to scan the stream data once and perform aggregation in the corresponding leaves based on Theorems 3.2 and 3.3, whose correctness has been proved. Step 2 needs to have one local H-header table for each level, and there are usually only a small number of levels (such as 6 in Figure 7). Only the exception cells take additional space. The space usage is small. Moreover, the H-cubing algorithm is fast based on the study in [18]. ■

Algorithm 2 (Popular-path) Compute regressions from the m- to o- layers following a "popular-path".

Input and Output are the same as Algorithm 1[7] except that the popular drilling path is given (e.g., the given popular path, $\langle (A_1, C_1) \to B_1 \to B_2 \to A_2 \to C_2\rangle$, is shown as the dark-line path in Figure 6).

Method.

[6] There are some subtle differences between the H-cubing here and the one described in [18] because the previous cubing does not handle multiple levels in the same dimension attribute. For lack of space, this will not be addressed further here.

[7] Notice that Algorithm 1 computes more exception cells than Algorithm 2 (i.e., more than necessary), since the former computes all the exception cells in every required cuboid, while the latter only computes (recursively) the exception cells' exception children, starting at the o-layer.

1. The same as Step 1 of Algorithm 1, except the H-tree should be constructed in the same order as the popular path (e.g., it should be $\langle (A_1, C_1) \to B_1 \to B_2 \to A_2 \to C_2 \rangle$).
2. Compute aggregation by rolling up the m-layer to the o-layer, along the path, with the aggregated regression points stored in the nonleaf nodes in the H-tree, and output only the exception cells.
3. Starting at the o-layer, drill on the exception cells at the current cuboid down to noncomputed cuboids, and compute the aggregation by rolling up a computed cuboid residing at the closest lower level. Output only the exception cells derived in the current computation. The process proceeds recursively until it reaches the m-layer.

Analysis. The H-tree ordering in Step 1 based on the drilling path facilitates the computation and storage of the cuboids along the path. Step 2 computes aggregates along the drilling path from the m-layer to the o-layer. At Step 3 drilling is done only on the exception cells and it takes advantage of the closest low level computed cuboids in aggregation. Thus the cells to be computed are related only to the exception cells, and the cuboids used are associated with the H-tree, which costs only minimal space overhead. ■

In comparison of the two algorithms, Algorithms 1 and 2, one can see that both need to scan the stream data only once. Regarding space, the former needs (1) one H-tree with the regression points saved only at the leaf and (2) a set of local H-header tables; whereas the latter does not need local H-headers, but it needs to store the regression points in both leaf and nonleaf nodes. Regarding computation, the former needs to compute all the cells although only the exception cells are retained but it uses precomputed results quite effectively. In comparison, the latter computes all the cells for the cuboids along the path only, but computes only the exception cells at each lower level, which may lead to less computation; however, it may not use intermediate computation results as effectively as the former. From this analysis, one can see that the two algorithms are quite competitive in both space and computation time and a performance study is needed to show their relative strength.

4.5 Making the algorithms on-line

For simplicity, so far we have not discussed how our algorithms deal with the "always-grow" nature of time-series stream data in an "on-line", continuously growing manner. We discuss this issue here.

The process is essentially an incremental computation method illustrated below, using the tilt time frame of Figure 4. Assuming that the memory contains the previously computed m and o-layers (plus the cuboids along the popular path in the case of using the popular-path *algorithm*), and the stream data arrive every minute. The new stream data are accumulated (by regression aggregation) in the corresponding H-tree leaf nodes. Since the time granularity of the m-layer is quarter, the aggregated data will trigger the cube computation once every 15 minutes, which rolls up from leaf to the higher level cuboids. When reaching a cuboid whose time granularity is hour, the rolled regression information (i.e., ISB) remains in the corresponding quarter slot until it reaches the full hour (i.e., 4 quarters), and then it rolls up to even higer levels.

Notice in this process, regression data in the time interval of each cuboid will be accumulated and promoted to the corresponding coarser time granularity, when the accumulated data reaches the corresponding time boundary. For example, the regression information of every four quarters will be aggregated to one hour and be promoted to the hour slot, and in the mean time, the quarter slots will still retain sufficient information for quarter-based regression analysis. This design ensures that although the stream data flows in-and-out, regression always keeps up to the most recent granularity time unit at each layer.

5 Performance Study

To evaluate the effectiveness and efficiency of our proposed algorithms, we performed an extensive performance study[8] on synthetic datasets. Limited by space, in this section, we report only the results on several synthetic datasets[9] with $100,000$ merged (i.e., m-layer) data streams, each consisting of a varied number of dimensions and levels. The results are consistent in other data sets.

The datasets are generated by a data generator similar in spirit to the IBM data generator [4] designed for testing data mining algorithms. The convention for the data sets is as follows: $D3L3C10T100K$ means that there are 3 dimensions, each dimension contains 3 levels (from the m-layer to the o-layer, inclusive), the node fan-out factor (cardinality) is 10 (i.e., 10 children per node), and there are in total 100K merged m-layer tuples.

All experiments were performed on a 750MHz AMD PC with 512 megabytes main memory, running Microsoft Windows-2000 Server. All methods were implemented using Microsoft Visual C++ 6.0. We compare the performance of the two algorithms as follows.

Our design framework has some obvious performance advantages over its alternatives in some aspects. Consequently, we do not conduct experimental performance study on these aspects (otherwise we will be comparing clear winners against obvious losers).

[8] Since regression analysis of time series in data cubes is a novel subject and there are no existing methods for this problem, we cannot compare our algorithms against previous methods.

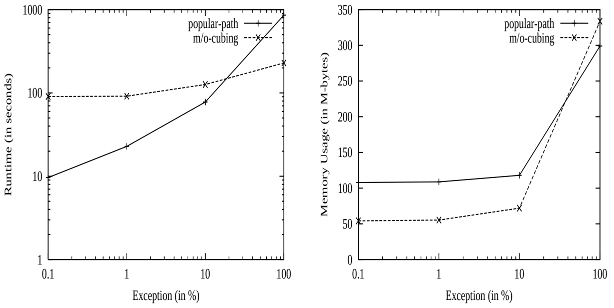
[9] We are also seeking for some industry time-series data sets for future study on real data sets.

These aspects include (1) *tilt time frame* vs. *full non-tilt time frame*, (2) *using minimal interesting layer* vs. *examining stream data at the raw data layer*, and (3) *computing the cube up to the apex layer* vs. *computing it up to the observation layer*.

Since a data analyst needs fast on-line response, and both space and time are critical in processing, we examine both time and space consumption in our performance study.

We have examined the following factors in our performance study: (1) time and space w.r.t. the exception threshold, i.e., the percentage of aggregated cells that belong to exception cells; (2) time and space w.r.t. the size (i.e., the number of tuples) at the m-layer; and (3) time and space w.r.t. the varied number of (dimensions and) levels.

The performance results are reported in Fig 8—10.

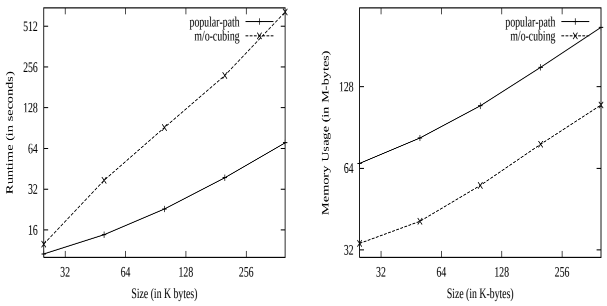


a) Time vs. exception b) Space vs. exception

Figure 8: Processing time and memory usage vs. percentage of exception (data set: $D3L3C10T100K$)

Figure 8 shows processing time and memory usage vs. percentage of exception, with the data set fixed as $D3L3C10T100K$. Since m/o-cubing computes all the cells from the m-layer all the way to the o-layer, despite the rate of exception, the total processing time is just slightly higher at high exception rate (i.e., when most of the cells are exceptional ones) than at low exception one. However, the memory usage will change a lot when exception rate grows since only the exception cells are retained in memory in this algorithm. On the other hand, popular-path computes only the popular path plus the drilling-down exception cells; when the exception rate is low, the computation cost is low, but when the exception rate grows, it costs more in computation time since it does not explore sharing processing as nicely as m/o-cubing. Moreover, its space usage is more stable at low exception rate since it take more space to store the cells along the popular path even when the exception rate is very low.

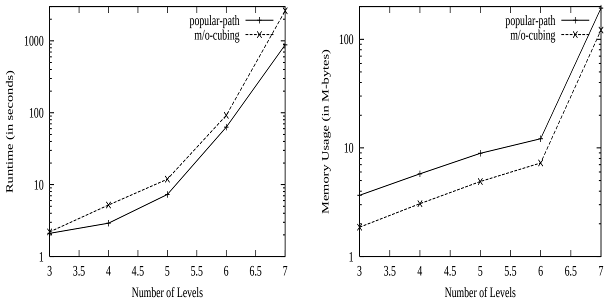
Figure 9 shows the processing time and memory usage vs. the size of the m-layer, with the cube structure of $D3L3C10$ and the exception rate at 1%, where the data sets with varied sizes are appropriate subsets of the same 100K data set. When the data size grows, popular-path is more scalable than m/o-cubing w.r.t. running time because m/o-cubing



a) Time vs. m-layer size b) Space vs. m-layer size

Figure 9: Processing time and memory usage vs. size of the m-layer (with cube structure of $D3L3C10$ and the exception rate of 1%)

computes all the cells between the two critical layers whereas popular-path computes only the cells along popular path plus a relatively small number of exception cells. However, popular-path takes more memory space than m/o-cubing because all the cells along the popular path needs to be retained in memory (which will be used in computation of exception cells of other cuboids).



a) Time vs. # of levels b) Space vs. # of levels

Figure 10: Processing time and space vs. # of levels from m- to o- layers (with the cube structure of $D2C10T10K$ and the exception rate at 1%)

Figure 10 shows the processing time and space usage vs. the number of levels from m- to o- layers, with cube structure of $D2C10T10K$ and the exception rate at 1%. In both algorithms, with the growth of number of levels in the data cube, both processing time and space usage grow exponentially. It is expected this exponential growth will be even more serious if the number of dimension (D) grows. This is one more verification of the "curse of dimensionality". Fortunately, most practical applications in time-series analysis (such as power consumption analysis) may involve only a small number of dimensions. Thus the method derived here should be practically applicable.

From this study, one can see that both m/o-cubing and popular-path are efficient and practically interesting algorithms for computing multi-dimensional regressions for time-series stream data. The choice of which one should be dependent on the expected exception ratio, the total (main) memory size, the desired response time, and how computing exception cells along a fixed path fits the needs of the application.

Finally, we note that this performance study deals with computing the cubes for the whole set of available stream data (such as 100K tuples). In stream data applications, it is likely that one just need to incrementally compute the newly generated stream data. In this case, the computation time should be substantially shorter than shown here although the total memory usage may not reduce much due to the need to store data in two critical layers, in popular path, as well as the retaining of exception cells.

6 Discussion

In this section, we compare our study with the related work and discuss some possible extensions.

6.1 Related work

Our work is related to: 1) mathematical foundations and tools for time series analysis, 2) similarity search and data mining on time series data, 3) on-line analytical processing and mining in data cubes, and 4) research into management and mining of stream data. We briefly review previous research in these areas and point out the differences from our work.

Statistical time series analysis is a well-studied and mature field [8], and many commercial statistical tools capable of time series analysis are available, including SAS, SPlus, Matlab, and BMDP. A common assumption of these studies and tools, however, is that users are responsible to choose the time series to be analyzed, including the scope of the object of the time series and the level of granularity. These studies and tools (including longitudinal studies [10]) do not provide the capabilities of relating the time series to the associated multi-dimensional multi-level characteristics, and they do not provide adequate support for on-line analytical processing and mining of the time series. In contrast, the framework established in this paper provides efficient support to help users form, select, analyze, and mine time series in a multi-dimensional and multi-level manner.

Similarity search and efficient retrieval of time series has been the major focus for time series-related research in the database community, such as [11, 2, 24, 25]. Previous data mining research also paid attention to time series data, including shape-based patterns [3], representative trends [22], periodicity [17], and using time warping for data mining [23]. Again, these[10] do not relate the multi-dimensional, multi-level characteristics with time series and do not seriously consider the aggregation of time series.

In data warehousing and OLAP, much progress has been made on the efficient support of standard and advanced OLAP queries in data cubes, including selective cube materialization [19], iceberg cubing [5, 18], cube gradients analysis [21, 9], exception [26], and intelligent roll-up [27]. However, the measures studied in OLAP systems are usually single values, and previous studies do not consider the support for regression of time series. In contrast, our work considers complex measures in the form of time series and studies OLAP and mining over time series data cubes.

Recently, several research papers have been published on the management and querying of stream data [6, 14, 15, 13], and data mining (classification and clustering) on stream data [20, 16]. However, these works do not consider regression analysis on stream data.

Thus this study sets a new direction: *extending data cube technology for multi-dimensional regression analysis*, especially for the analysis of stream data. This is a promising direction with many applications.

6.2 Possible extensions

So far we only considered two types of aggregation in time series data cubes, namely aggregation over a standard dimension, and that over the time dimension by merging small time intervals into larger ones. We note here that there is a third type of aggregation needed in a time series data cube: *aggregation over the time dimension obtained by* folding *small time intervals at a lower level in the time hierarchy to a higher level one.* For example, starting with 12 time series at the daily level for the 12 months of a year, we may want to combine them into one, for the whole year, at the monthly level. This folds the 365 daily values into 12 monthly values. Different SQL aggregation functions can be used for folding, such as sum, avg, min, max, or last (e.g., stock closing value). With minor extension of the methods presented here, such time dimension can be handled efficiently as well.

This study has been focused on multiple dimensional analysis of stream data. However, the framework so constructed, including tilt time dimension, monitoring the change of patterns in a large data cube using an m-layer and an o-layer, and paying special attentions on exception cells, is applicable to the analysis of nonstream time-series data as well.

Moreover, the results of this study can also be generalized to multiple linear regression to situations where one needs to consider more than one regression variable, for example when there are spatial variables in addition to a temporal variable. There is a need for multiple linear regression for multiple regression variable applications. For example, for environmental monitoring and weather forecast, there are frequently networks of sensors placed at different geographic locations. These sensors collect measurements at fixed time intervals. One may wish do regression not only on the time dimension, but also the three spatial dimensions. More generally, one may also include other

[10] `http://www.cs.ucr.edu/~eamonn/TSDMA/bib.html` is a fairly comprehensive time series data mining bibliography.

variables, such as human characteristics, etc. as regression variables.

It is worth noting that we have developed a general theory for this direction in the full paper. This theory is applicable to regression analysis using nonlinear functions, such as the log function, polynomial functions, and exponential functions.

7 Conclusions

In this paper, we have investigated issues for on-line, multi-dimensional regression analysis of time-series stream data and discovered some interesting mathematical properties of regression aggregation so that only a small number of numerical values instead of the complete stream of data need to be registered for multi-dimensional analysis. Based on these properties, a multi-dimensional on-line analysis framework is proposed, that uses *tilt time frame*, explores *minimal interesting and observation layers*, and adopts an *exception-based computation method*. Such a framework leads to two efficient cubing methods, whose efficiency and effectiveness have been demonstrated by our experiments.

We believe this study is the first one which explores on-line, multi-dimensional regression analysis of time-series stream data. There are a lot of issues to be explored further. For example, we have implemented and studied an H-cubing-based algorithm for computing regression-based data cubes. It is interesting to explore other cubing techniques, such as multiway array aggregation [28], and BUC [5], for regression cubing, as well as developing efficient algorithms for non-exception-driven computation algorithms. Moreover, we believe that a very important direction is to further develop data cube technology to cover additional, sophisticated statistical analysis operations, that may bring new computation power and user flexibility to on-line, multi-dimensional statistical analysis.

References

[1] S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. Naughton, R. Ramakrishnan, S. Sarawagi. On the computation of multidimensional aggregates. *VLDB'96*.

[2] R. Agrawal, K.-I. Lin, H.S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. *VLDB'95*.

[3] R. Agrawal, G. Psaila, E. L. Wimmers, and M. Zait. Querying shapes of histories. *VLDB'95*.

[4] R. Agrawal and R. Srikant. Mining sequential patterns. *ICDE'95*.

[5] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. *SIGMOD'99*.

[6] S. Babu and J. Widom. Continuous queries over data streams. *SIGMOD Record*, 30:109–120, 2001.

[7] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Record*, 26:65–74, 1997.

[8] D. Cook and S. Weisberg. *Applied Regression Including Computing and Graphics*. John Wiley, 1999.

[9] G. Dong, J. Han, J. Lam, J. Pei, and K. Wang. Mining multi-dimensional constrained gradients in data cubes. *VLDB'01*.

[10] P. Diggle, K. Liang, and S. Zeger. *Analysis of Longitudinal Data*. Oxford Science Publications, 1994.

[11] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. *SIGMOD'94*.

[12] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. *Data Mining and Knowledge Discovery*, 1:29–54, 1997.

[13] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. *SIGMOD'01*.

[14] A. Gilbert, Y. Kotidis, S. Muthukrishnan, M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. *VLDB'01*.

[15] J. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continuous data streams. *SIGMOD'01*.

[16] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. *FOCS'00*.

[17] J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. *ICDE'99*.

[18] J. Han, J. Pei, G. Dong, and K. Wang. Efficient computation of iceberg cubes with complex measures. *SIGMOD'01*.

[19] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. *SIGMOD'96*.

[20] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. *KDD'01*.

[21] T. Imielinski, L. Khachiyan, and A. Abdulghani. Cubegrades: Generalizing association rules. Rutgers University, Aug. 2000.

[22] P. Indyk, N. Koudas, and S. Muthukrishnan. Identifying representative trends in massive time series data sets using sketches. *VLDB'00*.

[23] E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping to massive dataset. *PKDD'99*.

[24] T. Kahveci and A. K. Singh. Variable length queries for time series data. *ICDE'01*.

[25] Y.-S. Moon, K.-Y. Whang, and W.-K. Loh. Duality-based subsequence matching in time-series databases. *ICDE'01*.

[26] S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of OLAP data cubes. *EDBT'98*.

[27] G. Sathe and S. Sarawagi. Intelligent rollups in multidimensional OLAP data. *VLDB'01*.

[28] Y. Zhao, P. M. Deshpande, and J. F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. *SIGMOD'97*.