# Mining Approximate Top-K Subspace Anomalies in Multi-Dimensional Time-Series Data[*]

Xiaolei Li Jiawei Han
University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA

## ABSTRACT

Market analysis is a representative data analysis process with many applications. In such an analysis, critical numerical measures, such as profit and sales, fluctuate over time and form time-series data. Moreover, the time series data correspond to market segments, which are described by a set of attributes, such as age, gender, education, income level, and product-category, that form a multi-dimensional structure. To better understand market dynamics and predict future trends, it is crucial to study the dynamics of time-series in *multi-dimensional* market segments. This is a topic that has been largely ignored in time series and data cube research.

In this study, we examine the issues of anomaly detection in multi-dimensional time-series data. We propose *time-series data cube* to capture the multi-dimensional space formed by the attribute structure. This facilitates the detection of anomalies based on expected values derived from higher level, "more general" time-series. Anomaly detection in a time-series data cube poses computational challenges, especially for high-dimensional, large data sets. To this end, we also propose an efficient search algorithm to iteratively select subspaces in the original high-dimensional space and detect anomalies within each one. Our experiments with both synthetic and real-world data demonstrate the effectiveness and efficiency of the proposed solution.

## 1. INTRODUCTION

Time series is the topic of study in numerous disciplines because it is the source of information in many applications. For example, temperature recordings on the space shuttle inform scientists of potential problems, stock value fluctuations notify financial analysts of potential gains or losses, and product purchase amounts disclose complex market dynamics. With such data, many interesting problems and solutions have been proposed, such as time-series indexing, periodicity detection, clustering, classification, and similarity search [13, 10, 14, 25].

An important problem in decision support is anomaly detection, which aims to find time series that are "unusual" within some population. While there have been studies on this topic [14], they often pose the input data as a flat set of time series. That is, abnormality of a single time series is defined in relation to others without much background knowledge. However, in many applications, there are descriptive attributes associated with the data.

EXAMPLE 1. *Figure 1 shows historical stock performances of Intel, Apple, and the NASDAQ Computers Index (an "average" of Intel, Apple and 557 other computer companies). All three stocks have "Computer" in their "Sector" attribute and the Index is an aggregate of all "Computer" stocks. We observe that Apple's stock had a significant upward trend for the second half of 2005 while the other two stayed flat. And from 2006 until the present, Intel's gains are always a constant lower than the index.* ■

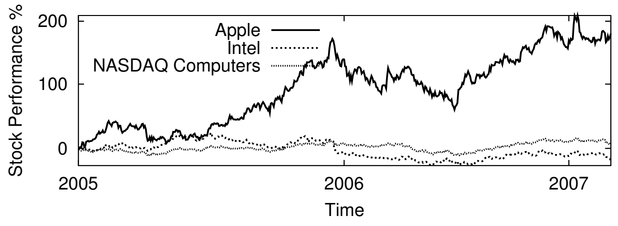


**Figure 1: Stock price time series**

Example 1 illustrates two anomaly types: trend and magnitude. More are formally defined later. It also shows how attributes attached to the time series can guide the comparison process. In this instance, the computer sector has been outperformed by one of its members (Apple) by a large margin. If the NASDAQ Computer Index were further divided into software, hardware, services, etc., one can further analyze comparisons with them. Example 2 shows a similar problem in a different domain.

EXAMPLE 2. *A marketing analyst is examining sales data from a store like WalMart. For every sale in the past 5 years, segment data on both the product and the purchaser*

[*]The work was supported in part by the U.S. National Science Foundation NSF IIS-05-13678/06-42771 and NSF BDI-05-15813. Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of the funding agencies.

*have been recorded. For the product, information such as category, store location, and price are available internally. For the purchaser, information such as age, gender, race, and income are obtained from store membership cards or credit cards. Within this vast amount of data, the analyst is searching for market segments (covering both customer and product dimensions) that have abnormal behavior. Suppose the analyst is particularly interested in males in the 18–35 age group. It is desirable that this search can automatically find anomalies like:* the sales of sports apparel in the past year to one subgroup, *males in the same age group who are married and have at least 1 child*, have been declining even though overall group sales have been increasing. ■

The above example shows a typical market analysis problem: Find anomalies in the market time-series data, associated with a set of attributes (*i.e.*, a market segment). Such a problem can be modeled by an OLAP (On Line Analytical Processing) framework [24]. One or a set of sales transactions sharing the same interested set of dimensions can be represented as a tuple in a multi-dimensional **fact table**, with the sales time series as a **measure**. A **data cube** can be constructed from the fact table by computing all the possible market segments and their aggregate sales time series. Then the question becomes how to find nontrivial anomalies in such a market **time-series data cube**.

This paper aims to discover nontrivial anomalies in a time-series data cube given some query. A brute force solution can be realized if the number of attributes (dimensions) is small. However, this would encounter major challenges in many real-world problems, where the data is high dimensional (such as ∼100). It is impossible to materialize a full cube in a high-dimensional space. Moreover, because anomaly in general does not have monotonic property [2], pruning in this large space is difficult. Further, if the time series in the data are of long sequences, computation over all of them in a cube would be expensive, too.

In this paper, we address this difficult problem via a divide-and-conquer approach. We make two main contributions. First, the high-dimensional data is partitioned automatically to discover interesting subsets of dimensions *and* tuples. Each subset forms a time-series data cube in itself, and we further propose an efficient top-$k$ cube anomaly mining algorithm. The combination of these two techniques leads to an efficient discovery of the global top-$k$ cube anomalies.

More specifically, this paper proposes an iterative subspace search algorithm named **SUITS** (Subspace Iterative Time-Series Anomaly Search) to mine top-$k$ anomaly cells. Given a *query probe cell* in a data cube, one would expect that a descendant cell, which is a subset, should roughly follow a similar evolutionary behavior. This leads to the computation of an *expected time series* and also the *anomaly measure*, which measures the difference between the expected and observed time series. Descendant cells of the probe cell are partitioned by their anomaly type and amount. For each partition, a correlated subspace data cube in the original high-dimensional space is extracted, and efficient top-$k$ anomaly mining is performed on it. This process iterates for all partitions. Each partition produces a local top-$k$, and they merge to form an approximation of the global top-$k$.

The rest of the paper is organized as follows. Section 2 defines the problem. Section 3 introduces the algorithm. Experiments are shown in Section 4. Related work is considered in Section 5 and the study concludes in Section 6.

## 2. PROBLEM DEFINITION

### 2.1 Preliminaries

**Time Series.** A time series $s(t)$ is a sequence or function which maps time values, $t$, to numerical values. The range of $t$ is usually restricted to some interval, and they are typically discrete values. A time series can represent any type of temporal data. For instance, the sequence $s(t) = \langle 5, 10, 13, 7, 2\rangle$ could represent the daily sales of televisions at a store over a 5-day interval: $t = [0, 5]$.

**Data Cube.** Given a relation $R$, a *data cube* (denoted as $C_R$) is the set of aggregates from all possible group-by's on $R$. In an $n$-dimensional data cube, a cell $c = (a_1, a_2, \ldots, a_n : m)$ (where $m$ is the cube measure) is called a $k$-dimensional group-by cell (*i.e.*, a cell in a $k$-dimensional cuboid) if and only if there are $k$ ($k \le n$) values among $(a_1, a_2, \ldots, a_n)$ which are not $*$ (*i.e.*, all). Given two cells $c_1$ and $c_2$, let $V_1$ and $V_2$ represent the set of values among their respective $(a_1, a_2, \ldots, a_n)$ which are not $*$. $c_1$ is the *ancestor* of $c_2$ and $c_2$ is a *descendant* of $c_1$ if $V_1 \subset V_2$. $c_1$ is the *parent* of $c_2$ and $c_2$ is a *child* of $c_1$ if $V_1 \subset V_2$ and $|V_1| = |V_2| - 1$. These relationships also extend to cuboids and form a structure called the *cuboid lattice*. An example is shown in Figure 2. The “All” or *apex* cuboid holds a single cell where all its values among $(a_1, a_2, \ldots, a_n)$ are $*$. On the other extreme, the *base* cuboid at the top holds cells where none of its $(a_1, a_2, \ldots, a_n)$ values is $*$.

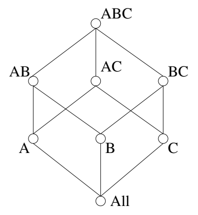


**Figure 2: Cuboid lattice**

**Input Data.** Consider a relation $R$ with $n$ attributes $A_1$, $A_2$, …, $A_n$. Each attribute $A_i$ contains discrete values. Let there be $t$ tuples in this relation with transaction IDs of $tid_1, tid_2, \ldots, tid_t$. Let there also be a set of time series $S = \{s_1, s_2, \ldots, s_t\}$ where $s_i$ is associated with tuple $tid_i$.

| Gender | Education | Income | Product | Profit | Count |
|---|---|---|---|---|---|
| Female | Highschool | 35k–45k | Food | $s_1$ | $u_1$ |
| Female | Highschool | 45k–60k | Apparel | $s_2$ | $u_2$ |
| Female | College | 35k–45k | Apparel | $s_3$ | $u_3$ |
| Female | College | 35k–45k | Book | $s_4$ | $u_4$ |
| Female | College | 45k–60k | Apparel | $s_5$ | $u_5$ |
| Female | Graduate | 45k–60k | Apparel | $s_6$ | $u_6$ |
| Male | Highschool | 35k–45k | Apparel | $s_7$ | $u_7$ |
| Male | College | 35k–45k | Food | $s_8$ | $u_8$ |

**Table 1: Input market segment data**

A market analysis sample data is shown in Table 1. Each of the four $A_i$'s represents an attribute on either the customer or the product. Each tuple in this relation corresponds to a market segment, and the associated time series measures the "Profit" over time. For all $s_i$ in $S$, the period and sampling rate are assumed to be the same (otherwise, preprocessing and normalization can be performed). In addition, count $u_i$ is associated with each market segment. It records the number of records (*i.e.*, people) in that segment, and its value is initialized to 1 by default if the field was originally nonexistent.

## 2.2 The Anomaly Search Problem

The problem to be studied is as presented in Example 2. That is, given a relation $R$ and its associated time-series set $S$, a probe cell $p \in C_R$, and an anomaly function $g$, find the **anomaly cells among descendants of $p$ in $C_R$ as measured by $g$**. To search all data and perform a global analysis, one can simply set $p$ to empty; the rest of the algorithm remains unchanged.

To make the algorithm more practical, we add three extra conditions: (1) each abnormal cell must satisfy a minimum count (support) threshold, which eliminates trivially small market segments, (2) anomalies do not have to hold for the entire time series, *i.e.*, $g$ can be applied to sub-sequences, and (3) only the **top $k$ anomaly cells as ranked by $g$** are returned. These conditions better match usage habits of analysts but are not critical to the core algorithm. For example, instead of condition (3), one may ask for all the anomalies larger than a fixed threshold.

Notice that although $p$ is a cube cell, its description is like a selection query. In Example 2, $p$ = (Gender = "Male", Age = "18–35") and all other dimensions are set to $*$. This can be treated as a selection query, $\sigma_p(R)$, *i.e.*, select exactly the set of tuples in $R$ which satisfy $p$ in $C_R$.

For each cell $c$ in $C_R$, there is an associated time series, denoted as $s_c$, and called an **observed time series**. In the context of a query probe $p$, $s_c$ is computed by aggregating time series from $S$ whose corresponding tid's are in $\sigma_p(R)$ and also $c$.

$$s_c = \sum_{tid_i \in\ c\ \cap\ \sigma_p(R)} s_i \tag{1}$$

In market analysis, $S$ is typically a numerical measure, such as sales or profit. The aggregation function can be either SUM or AVG, depending on the application semantics. Here we take SUM by default. In general, any distributive or algebraic aggregation function may be substituted.

At each cell, we will also calculate an **expected time series**, denoted by $\hat{s}_c$. The measure of anomaly will be a function on the observed and expected time series, *i.e.*, $g(s_c, \hat{s}_c) \to \mathbb{R}$.

## 2.3 Expected Time Series

Semantically, each cell in the data cube is a market segment, and so is the **probe cell** or **probe segment** $p$. Since the query seeks anomalies in $p$'s descendant cells (which are sub-segments inside the probe segment), the expected time series of any descendant cell of $p$ is computed from the probe cell $p$. A simple example below gives the intuition.

EXAMPLE 3. *Let $R$ be the relation shown in Table 1, and $p$ be (Gender = "Female", Product = "Apparel", Education = $*$, Income = $*$). Table 2 shows $p$ and all of $p$'s descendant cells. Notice that the Gender and Product dimensions have been dropped since they are the same for all tuples. The Count attribute has been filled with the sample values.* ∎

| $c$ | | $s_c$ | $\|c\|$ |
|---|---|---|---|
| Education | Income | Profit (time-series) | Count |
| $*$ | $*$ | $s_2 + s_3 + s_5 + s_6$ | 1000 |
| Highschool | $*$ | $s_2$ | 150 |
| College | $*$ | $s_3 + s_5$ | 800 |
| Graduate | $*$ | $s_6$ | 50 |
| $*$ | 35k–45k | $s_3$ | 200 |
| $*$ | 45k–60k | $s_2 + s_5 + s_6$ | 800 |
| Highschool | 45k–60k | $s_2$ | 150 |
| College | 35k–45k | $s_3$ | 200 |
| College | 45k–60k | $s_5$ | 600 |
| Graduate | 45k–60k | $s_6$ | 50 |

**Table 2: $p$ = ("Female", "Apparel")**

Every cell in the data cube has an associated measure and count. At the probe cell, we will term them the **probe measure** and the **probe count**, denoted by $s_p$ and $|p|$. In this example, $s_p = s_2 + s_3 + s_5 + s_6$ and $|p| = 150 + 200 + 600 + 50 = 1000$.

One of the descendant cells of $p$ is ($Education$ = "$College$", $Income$ = $*$), which has an observed time series of $(s_3 + s_5)$ and count of 800. In a market where a segment behaves proportionally to its size, one would expect this segment to be responsible for 80% (800/1000) of the probe profit. In other words, $0.8 \times s_p$. Similarly, the expected time series of ($Education$ = "$College$", $Income$ = "$45k - 60k$") would be 60% of the probe profit.

Let $c$ be a cell in a data cube, and $|c|$ be the count of $c$ within segment $p$. $|c|$ is computed by aggregating the counts of tuples in $\sigma_p(R)$ that contribute to $c$:

$$|c| = \sum_{tid_i \in\ c\ \cap\ \sigma_p(R)} u_i \tag{2}$$

In general, $u_i$, $|c|$, and $|p|$ could also be time series since counts can also change over time. The **expected time series**, $\hat{s}_c$, is defined as:

$$\hat{s}_c = \left(\frac{|c|}{|p|}\right) s_p \tag{3}$$

Intuitively, the expected time series makes the assumption that a market segment behaves *proportionally to the probe segment* according to its size.

## 2.4 Anomaly Definition

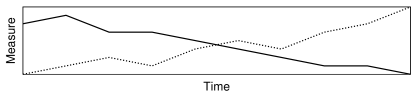
As mentioned previously, $g(s_c, \hat{s}_c)$ or simply $g$ measures the anomaly at each cell. A basic $g$ could be the Euclidean distance between $s_c$ and $\hat{s}_c$. However, this is rather uninformative to analysts. In market analysis, there are usually semantics associated with behavioral anomalies. Below, we list four primary types although they can be extended and refined based on the application. Figure 3 shows sample illustrations where dotted lines are expectations and solid lines are observations.

1. **Trend**: A trend anomaly occurs when the expected overall trend of a segment is different from the observed.

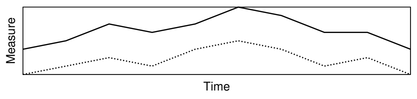
2. **Magnitude**: A magnitude anomaly occurs when a segment's measure is significantly more or less than expected but the overall trend is the same.

3. **Phase**: A phase anomaly occurs when a segment's observed behavior is significantly ahead of or behind the expected behavior in terms of time. The trend and magnitude are similar though.

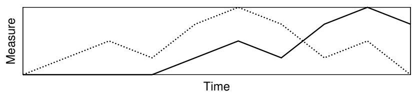
4. **Misc**: Miscellaneous anomalies are the ones which do not match any of the three above.



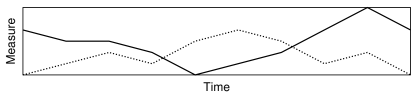
(a) Trend Anomaly



(b) Magnitude Anomaly



(c) Phase Anomaly



(d) Miscellaneous Anomaly

**Figure 3: Anomaly types**

### 2.4.1 Linear Regression

To represent and detect anomalies, we will use the **first-order linear regression**. Although simple, it is very effective at catching the big anomalies. Because of its limited representative power, small local anomalies are smoothed over and only the major trends are left. Alternatively, higher order regression or more sophisticated methods [14, 13] may be substituted to catch more subtle anomalies. Further, many business applications have natural partitionings of time series so our comparative analysis will be on the general trend of *sub-sequences*. As a result, the regression will only be performed on relatively short time series (*i.e.*, piece-wise regression). Lastly, as we will show later, aggregation in the data cube can rely exclusively on the regression parameters and bypass the original bulky time-series data [8].

An $l$-th order *polynomial fit* for a time series of $n$ observations: $\langle z(t_0), z(t_1), \ldots, z(t_n)\rangle$, where $z(t)$ is the underlying function, is an $l$-th order polynomial estimation function:

$$\hat{z}(t) = a_0 + a_1\, t + a_2\, t^2 + \ldots + a_k\, t^l \qquad (4)$$

where $\hat{z}(t)$ is the estimated value of $z(t)$. If we set $l = 1$, $\hat{z}(t)$ is then a line of the form $a_0 + a_1\, t$ where $a_0$ is commonly known as the *y-intercept* and $a_1$ as the *slope*. $a_0$ and $a_1$ can be calculated directly via the least squares error fitting.

### 2.4.2 Measuring Trend Anomaly

A trend anomaly indicates a difference in general market trends, as shown in Figure 3(a). The slope of the regression line naturally captures this. Let $a_1$ be the slope of the observed time series and $\hat{a}_1$ be the slope of the expected time series. The difference between $a_1$ and $\hat{a}_1$ then measures the **trend anomaly**.

$$g_{trend}(s_c, \hat{s}_c) = a_1 - \hat{a}_1$$

### 2.4.3 Measuring Magnitude Anomaly

A magnitude anomaly indicates a difference in the amount of the measure, as shown in Figure 3(b). The $y$-intercept of the regression line is a fitting representative of this. Let $a_0$ and $\hat{a}_0$ be the $y$-intercepts of the observed and expected time series respectively. The difference between them measures the **magnitude anomaly**; though when the trend anomaly is triggered, magnitude is ignored.

$$g_{magnitude}(s_c, \hat{s}_c) = a_0 - \hat{a}_0$$

### 2.4.4 Measuring Phase Anomaly

A phase anomaly indicates a lag or shift in time of the measure, as shown in Figure 3(c). The $x$-intercept of the regression line is able to capture this with some care. First, we check that trend anomaly is not triggered. Then, we notice that a difference in $x$-intercept is always paired with a difference in y-$sy$intercept, and vice-versa. In order to determine the anomaly type, we rely on the original data. Given the choice of magnitude or phase anomaly, we shift the observed time series either vertically or horizontally to offset the anomaly difference. Then, whichever shift produces the most similarity between the expected and observed time series is the answer. If neither produces sufficient similarity, we mark it as misc. anomaly. The exact parameters depend on the application. We implemented this mechanism for a major industry partner, with satisfactory performance. Let $b_0$ and $b_1$ be the $x$-intercepts of the observed and expected time series after regression respectively.

$$g_{phase}(s_c, \hat{s}_c) = b_0 - \hat{b}_0$$

### 2.4.5 Measuring Miscellaneous Anomaly

To measure **miscellaneous anomalies**, as shown in Figure 3(d), we use the Euclidean distance between $s_c$ and $\hat{s}_c$, denoted as $d(s_c, \hat{s}_c)$. Though some preprocessing might be required to align $s_c$ and $\hat{s}_c$ correctly. In order to reduce redundancies with the other anomaly types, miscellaneous anomaly is only applicable when the other three types are not triggered.

$$g_{misc}(s_c, \hat{s}_c) = d(s_c, \hat{s}_c)$$

## 2.5 Ranking Anomalies in Data Cube

With $g$ defined for the anomaly types, we can rank all descendant cells of $p$ in descending order according to their absolute $g$ values. In all four types, a larger absolute $g$ value indicates a more substantial anomaly. The original query would then return the top-$k$ market segments in this ranking. However, because the four types of $g$'s are incompatible, it may not make sense to rank them together. Rather, it is more sensible to have a separate ranking for each distinct $g$. As a result, the top-$k$ would be on individual types. With four types of anomalies defined, the final result would consist of $4k$ market segments.

Table 3 shows a summary of $g$ on all the anomaly types. An additional normalization or adjustment process is often needed to add weighting factors to $g$ depending on the applications. For example, one may like to use average instead of sum in comparison, which will need to divide the value by count. For some other applications, one may like to give more weight to the market segments that are larger (hence bigger count) to indicate the preference in analysis.

| Type | Function $g$ |
|---|---|
| Trend | $norm(a_1 - \hat{a}_1)$ |
| Magnitude | $norm(a_0 - \hat{a}_0)$ |
| Phase | $norm(b_0 - \hat{b}_0)$ |
| Misc | $norm(d(s_c, \hat{s}_c))$ |

**Table 3: Anomaly detection functions**

## 3. MINING TOP-K ANOMALIES IN DATA CUBES

With $g$ defined, we return to the original problem of finding top-$k$ anomaly cells among the descendants of $p$. A naïve solution to this problem is given in Algorithm 1. Its main observation is that $C_R$ is unnecessary because the query only focuses on $p$. Thus, it only computes the data cube $C_p$ using $\sigma_p(R)$ as the fact table. After $C_p$ is constructed, the top-$k$ anomaly cells within it are returned.

**Algorithm 1** Naïve Top-$k$ Anomalies

```
Input: Relation $R$, time-series data $S$, query probe cell $p$,
    anomaly function $g$, parameter $k$, minimum support $m$
Output: Top-$k$ scoring cells in $C_p$ as ranked by $g$ and
    satisfies $m$

1. Retrieve data for $\sigma_p(R)$
2. Compute the data cube $C_p$ with $\sigma_p(R)$ as the fact table
       with $m$ as the iceberg parameter
3. Return top $k$ anomaly cells in $C_p$ for each $g$
```

The core difficulty with Algorithm 1 is how to deal with the high dimensional space; if there are $n$ attributes in $R$, there are $2^n$ cuboids (subspaces) in $C_p$ to examine in order to produce the final answer. This effectively prohibits full materialization of $C_p$ for a medium $n$ even if $\sigma_p(R)$ does not contain many tuples. To solve this problem, we propose a new algorithm SUITS, which iteratively select subspaces with the most potential of containing a top-$k$ anomaly. Anomaly detection over a subspace tends to be very efficient since a subspace has typically a small number of attributes (dimensions). Fortunately, because anomalies are rare by definition, many of the $2^n$ subspaces are not correlated with anomalies. Figure 4 shows the general framework.

A natural question is then, "which subspaces out of the $2^n$ should one examine?" SUITS chooses them based on the behaviors of the time series data (*i.e.*, $t_i$'s). Roughly, abnormal time series in $\sigma_p(R)$ are separated into individual anomalies, and their correlated subspaces are chosen as **candidate subspaces**. These subspaces are then examined via exact cubing analysis. This approach avoids the curse of dimensionality in the original input data and turns it into a set of manageable sub-problems.

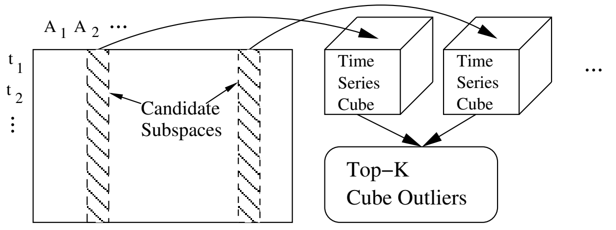


**Figure 4: SUITS Framework**

Additionally, during the computation of top-$k$'s within a single subspace, the search space can be pruned if one detects that certain cuboids and their descendants does not have the potential to penetrate the top-$k$. This pruning method is developed in SUITS, and unpromising lattices in the data cube are avoided.

In summary, the algorithm proceeds iteratively as follows: (1) search for a group of anomalies, (2) find a subspace correlated with the group, and (3) compute the local top-$k$ anomalies in the subspace data cube. The local top-$k$'s of step (3) are merged together to form the global top-$k$. Though this merge is an approximation, we will show empirically that it usually matches the true top-$k$.

### 3.1 Retrieving $\sigma_p(R)$

Much like the naïve algorithm, the new algorithm also needs to first retrieve the set of data relevant to the query probe $p$, *i.e.*, $\sigma_p(R)$. Since there will be many different query probes posed to the same database, it is important to make this retrieval and its subsequent processing efficient. Thus, we perform preprocessing by pre-computing and storing $C_R$'s shell-fragments [15] independent of the query and develop a shell fragment-based retrieval method.

A single *shell fragment* is a cuboid in $C_R$ on a $d$-dimensional attribute group where $d$ is a small number (*e.g.*, 1 to 3). For each cell in a fragment, the tid list of the associated tuples in $R$ is recorded. For example, the shell fragment for the Gender dimension would contain two cells (*i.e.*, "Male" and "Female") and each would record essentially an inverted index on the tid's. A complete set of shell fragments (*i.e.*, where each dimension in $R$ is represented in at least one shell fragment) is sufficient to compute any query on $C_R$. Shell fragments are efficient both in terms of speed and space.

Using these tid lists, retrieving $\sigma_p(R)$ at query time is simple. For each attribute-value restriction pair in $p$, we fetch its tid list from the most appropriate shell fragment. The intersection of all such tid lists is exactly $\sigma_p(R)$. This process is efficient no matter how many dimensions there are in $R$. Additionally, if $p$ overlaps with some multi-dimensional shell fragments, efficiency will be vastly improved since those intersections are already pre-computed.

### 3.2 Selecting Candidate Subspaces

The idea of examining subspaces also exists in other problems. Subspace clustering [19] aims to find clusters in some of the $2^n$ subspaces. Principle component analysis and singular value decomposition also find more useful subspaces. In these problems, useful subspaces are discovered using signals such as density or class labels. In SUITS, the time series data are the signals. Intuitively, a significant anomaly at a cube cell should carry through to some of its descendants;

for if all descendants are normal, their common ancestor would also be normal. Furthermore, descendants of common abnormal ancestors should also exhibit similar anomalies. Specifically, tuples in $\sigma_p(R)$ that share a common abnormal ancestor cell, which would also be a descendant cell of $p$, are likely to exhibit similar anomalies.

SUITS exploits this notion by *grouping* the tuples in $\sigma_p(R)$ based on their anomaly types and values. For a set of tuples in $\sigma_p(R)$ to be in the same group, they must have (1) the same anomaly type, (2) similar anomaly scores (*e.g.*, $\pm\delta$ range), and (3) same time span. Roughly, tuples within a group are base-level descendants of a single anomaly in a (possibly high level) descendant of the probe cell. At each iteration, the largest group is extracted and the most promising attribute values within it form the candidate subspace. This process starts with the construction of a **Time Anomaly Matrix**.

The Time Anomaly Matrix has size $T \times Q$, where $T$ is the number of tuples in $\sigma_p(R)$ and $Q$ is the number of subsequences SUITS will examine within $S$. Partitioning the time series satisfies the condition that anomalies will be found in sub-sequences. In business and many other applications, how to partition the $s_i$'s is often natural, *e.g.*, every financial quarter.

Each entry $(i, j)$ in the matrix corresponds to the $j$th time series sub-sequence of the $i$th tuple in $\sigma_p(R)$, and the value stored in the entry is the anomaly type and anomaly value of the corresponding sub-sequence. Formally, let $s[j]_{c_i}$ represent the $j$th sub-sequence of $s_{c_i}$ where $c_i$ is the $i$th cell or tuple in $\sigma_p(R)$. Then, the $(i, j)$ entry in the matrix contains the output of $g(s[j]_{c_i}, \hat{s}[j]_{c_i})$.

Example 4 (Time Anomaly Matrix). *Using R from Table 1, let p be (Gender = "Female", Product = "Apparel"). The last 4 rows in Table 2 show $\sigma_p(R)$. For each tuple's $s_i$, we divide it into three pieces. Table 4 shows the results. Under each piece is the anomaly type; we have omitted the score for simplicity.* ■

The Time Anomaly Matrix merely calculates anomalies in the base cuboid. Our overall goal is to find anomalies in possibly high-level cube cells. To enumerate all possible cube cells is cumbersome; instead we iteratively select potential subspaces. This process starts with the grouping of cells in the matrix. We use a simple method of hashing entries in the matrix into buckets by their anomaly type, anomaly score, and time. Each bucket will only hold one type, a small range of scores, and consecutive time spans. We then greedily select the largest such group by choosing the largest bucket. In Table 4, the six entries with Magnitude anomaly (in bold) form the largest group.

The next step is to find a useful subspace associated with this group. To exhaustively search for this is prohibitive. In many high-dimensional problems, greedy/heuristic methods are used to "bypass" the curse of dimensionality. For example, decision trees use an information-theoretic heuristic to greedily choose the decision nodes independently. CLIQUE [3] uses a coverage measure to select clusters in subspaces and greedily grow them to form bigger clusters. [2] uses an evolutionary algorithm to detect outliers in high-dimensional data. In SUITS, we take a similar approach by evaluating the attribute values individually in a statistical test to determine how well alone it correlates with the anomaly group. We term the score of this test the **Anomaly Likelihood** (AL) score. The few top-scoring values then form the correlated subspace. To measure the AL score of an attribute-value pair $(a_i = v_j)$, the purity of attribute $a_i$ is calculated first via *entropy*.

$$Entropy(a_i) = -\sum_{v_j \in a_i} p(v_j) \log_2 p(v_j) \tag{5}$$

A "pure" attribute, that is an attribute whose values are homogeneous, would have low entropy; while an "impure" attribute whose values are uniformly distributed would have high entropy. If an attribute is pure, it is more likely to be correlated with the group than one that is impure. To give a trivial example, consider the Gender attribute for $p$ in Example 4. It is 100% pure because all its values are "Female" and is trivially correlated with any anomaly. The equation below shows the AL score formula.

$$AL(a_i = v_j) = Frequency(a_i = v_j) \times Entropy(a_i)^{-1} \tag{6}$$

For each value $v_j$, it is also weighed by its frequency within the group. One can see that attribute values that occur very frequently and within a homogeneous attribute will have high AL scores.

Example 5 (AL Scores). *Table 5 shows results from the Magnitude anomaly group in Table 4. Within the Income attribute, the value "45k–60k" appears 3 times and no other value appears. The Income attribute is pure and thus scores an infinity for the AL score. Within the Education attribute, 3 different values appear uniformly, which maximizes entropy. In this case, Income = "45k–60k" is clearly correlated with the anomaly while Education is not. The AL score reflects this notion.*

| Attribute Value | Frequency | AL Score |
|---|---|---|
| Income = 45k–60k | 3 | $\infty$ |
| Education = Highschool | 1 | 1.58 |
| Education = College | 1 | 1.58 |
| Education = Graduate | 1 | 1.58 |

**Table 5: Attribute value AL scores**

In practice, Table 5 would be much bigger and the differences within it would not be as clear-cut. We select the top few scoring (5–7) attribute-value pairs to be **candidate attribute values**. The subspace formed by these values is the **candidate subspace**.

## 3.3 Discovering Top-K Anomaly Cells

The set of candidate attribute values describes a subspace within the original high-dimensional space. Its correlation to anomalies has only been suggested via simple entropy analysis. In this section, more exact cubing analysis is performed and the top-$k$ cells in the subspace are found.

Let the set of candidate attribute values be $B$. A straightforward solution is to materialize the data cube $C_B$, rank all cells by $g$, and return the top $k$. Note that the dimensionality of $C_B$ is not necessarily equal to $|B|$. In Table 5 for example, even if all four attribute values are added to $B$, dimensionality is still just two. Second, to compute $C_B$, all tuples in $\sigma_p(R)$ are used, not just the candidate group. This ensures the detected top-$k$ patterns apply globally. Though to ensure sub-sequences are searched, $C_B$ only includes the time span of the found group.

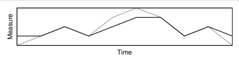
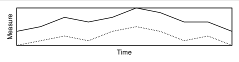
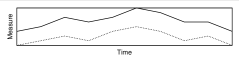
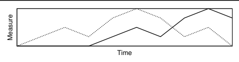
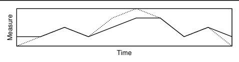
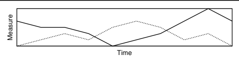
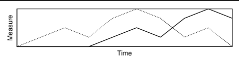
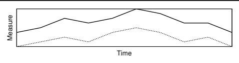
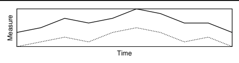
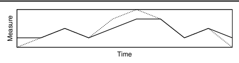
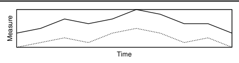
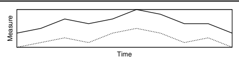
| Education | Income | $S[1]$ | $S[2]$ | $S[3]$ |
|---|---|---|---|---|
| Highschool | 45k–60k |  None |  **Magnitude** |  **Magnitude** |
| College | 35k–45k |  Phase |  None |  Misc |
| College | 45k–60k |  Phase |  **Magnitude** |  **Magnitude** |
| Graduate | 45k–60k |  None |  **Magnitude** |  **Magnitude** |

**Table 4: Time Anomaly Matrix**

This solution is definitely viable since $C_B$ is relatively small. But we make two modifications in order to improve its efficiency. First, the number of times regression needs to be performed can be reduced through a property of the least-square error fitting. Second, branches in $C_B$ can be pruned from the top-$k$ search via an upper bound on $g$. We explain both of these modifications below.

### 3.3.1 *Cube Aggregation*

Recall linear regression is used to represent $s_c$ and $\hat{s}_c$. To compute the regression parameters from the raw data at each cell in the cube can be costly: At each cell, one has to aggregate all the time series from its child cells and then find the least-square fit. Fortunately, it turns out that there is a shortcut.

Theorem 1 (Regression Aggregation). *Let there be $l$ time series $(s_{c1}, s_{c2}, \ldots, s_{cl})$ with their respective regression y-intercepts and slopes. The y-intercept is the sum of the $l$ individual y-intercepts: $a_0^{aggregate} = \sum_{i=1}^{l} a_0^i$, and the slope of the $l$ time series' aggregate is sum of the $l$ individual slopes: $a_1^{aggregate} = \sum_{i=1}^{l} a_1^i$.*

A proper proof is given in [8]. Theorem 1 means that for all non-base cuboid cells in the data cube, regression can be calculated via aggregation of regression parameters of their child cuboids as opposed to from the raw time series data. This alternative calculation is lossless with respect to the least-square fit. Since we are using first order linear regression, there would be only two parameters that each child cuboid has to pass up to their parent during aggregation. Comparing this with full aggregation of the raw time series, which could have hundreds of values in just one series, the improvement in efficiency is obvious.

Example 6 (Regression Aggregation). *Let $c_z$ be a high-level cell and $c_x$ and $c_y$ be its only two descendant cells. After regression on $c_x$ and $c_y$, let $a_0^x$ and $a_1^x$ be the regression parameters for $c_x$ and $a_0^x$ and $a_1^y$ be the same for $c_y$. By Theorem 1, the regression parameters for $c_z$, $a_0^z$ and $a_1^z$, are equal to the sums of the regression parameters at $c_x$ and $c_y$. In other words, $a_0^z = a_0^x + a_0^y$ and $a_1^z = a_1^x + a_1^y$.*

### 3.3.2 *Top-K Pruning*

Theorem 1 improves the speed of cubing but does not change the landscape of search. In order to reduce the search space, pruning during cubing is necessary. In SUITS, computing $C_B$ occurs in a bottom-up fashion [5] and is shown in Figure 5. That is, cubing starts at the apex and moves onto higher dimensional cells. Bottom-up cube computation has two advantages. First, it facilitates iceberg pruning: If a cell does not satisfy the minimum support condition, none of its descendants does and thus can all be pruned. Second, it starts at the most general cells, which, if $g$ is weighted by Count, are more likely to be the most abnormal cells.

As it turns out, at each cell, one can also calculate an **upper bound** on the $g$ scores of all descendant cells. This upper bound can be used to facilitate more efficient top-$k$ calculation. For example, suppose the lowest anomaly score in the top-$k$ seen so far in cubing is $x$. If the upper bound at a certain branch of the cube is strictly less than $x$, the entire branch can be pruned.

The calculation of upper bound is dependent on anomaly types. For Miscellaneous anomalies, the Euclidean distance has a natural upper bound by the triangle inequality: At any cell, the sum of the individual Euclidean distances of all cells which contribute to it is the upper bound. For Trend and Magnitude anomalies, we have the following Lemma.

Lemma 1 (Trend/Magnitude Bound). *At any cell $c$ in a data cube, let $\mathcal{A}$ be the set of anomaly values calculated by either $g_{trend}$ or $g_{magnitude}$ for all the tuples which contribute to $c$. Let $\mathcal{P}$ be the set of positive values in $\mathcal{A}$ and $\mathcal{N}$ be the set of negative values. $\mathcal{A} = \mathcal{P} \cup \mathcal{N} \cup \{0\}$. Also let $P$ be the sum of values in $\mathcal{P}$ and $N$ be the sum of values in $\mathcal{N}$. Then, $\max(|P|, |N|)$ is the absolute upper bound on $g$ for all descendants of $c$.*

**Proof**: First, by the definitions of $g_{trend}$ and $g_{magnitude}$ and Theorem 1, $g$ at cell $c$ can be calculated by simply adding up the individual values in $\mathcal{A}$. This holds because $g_{trend}$ and $g_{magnitude}$ are differences of sums of regression variables, which Theorem 1 shows can be aggregated losslessly. Second, let $c'$ be any descendant of $c$ and let $\mathcal{A}'$ be the set of anomaly values calculated by $g$ for tuples which contribute to $c'$. By definition of a data cube, tuples belonging to $c'$ are a subset of tuples belonging to $c$. Thus, $\mathcal{A}' \subset \mathcal{A}$. $P$ and $N$ represent the maximum positive and negative aggregates in $\mathcal{A}$. Thus any $\mathcal{A}'$ could never aggregate to an absolute $g$ value higher than $\max(|P|, |N|)$. ■

For the Phase anomaly, the upper bound is slightly trickier because the $x$-intercept is a fraction of regression values (*x-intercept* = −*slope*/*y-intercept*). In this case, one can derive a looser upper bound using similar ideas from Lemma 1 except by setting the denominator in $\mathcal{P}$ and $\mathcal{N}$ to the smallest value.

As an example, suppose there are three dimensions in a selected subspace. Here, we want to compute the top-3 trend anomaly cells in a 3D data cube. Table 6 shows the base cuboid. Sample differences between the observed and expected time series are shown. $a_1$ is the observed slope and $\hat{a}_1$ is the expected slope. The difference is a trend anomaly. Cubing starts at the apex and proceeds bottom-up; Figure 5 shows the ordering. The algorithm records a "current-best" top-$k$, denoted by $K$, which records the top-$k$ seen so far during cubing. It is initialized to an empty set. While $|K| < k$ or the upper bound value at any cell is larger than the smallest anomaly in $K$, the algorithm proceeds to compute the data cube.

| Age | Sex | Height | $a_1$ | $\hat{a}_1$ | $a_1 - \hat{a}_1$ |
|---|---|---|---|---|---|
| 5 | M | Short | 1 | 0 | 1 |
| 5 | M | Tall | 0 | 0 | 0 |
| 5 | F | Tall | −3 | −3 | 0 |
| 10 | F | Tall | 1 | 3 | −2 |
| 10 | F | Short | 2 | −3 | 5 |

**Table 6: Sample base cuboid**

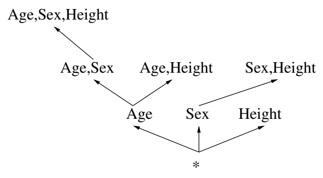


**Figure 5: Bottom-Up Cube Computation**

After calculating the apex cell, $K = \{(*, *, *) : 4\}$. Next, cuboid Age is calculated. The first two rows in Table 7 show cells in Age. For example, $a_1$ of $(5, *, *)$ is calculated from $1 + 0 - 3$. Because $|K| < 3$ at this point, both $(5, *, *) : 1$ and $(10, *, *) : 3$ are added to $K$. Next, cuboid (Age, Sex) is calculated. Rows 3–5 in Table 7 show the cells. The last column shows the upper bound. At $(10, F, *)$, the upper bound is 5, because 5 is the only positive anomaly value in Table 6 for $(10, F, *)$ and the negative value has a smaller absolute value ($|-2| < |5|$). At $(5, M, *)$ and $(5, F, *)$, the upper bounds are less than or equal to the smallest anomaly in $K$. Thus, their descendants, *e.g.*, $(5, M, Short)$, are pruned from future search. $(10, F, *)$ is added to $K$ and search recurses on its descendants.

| Age | Sex | Height | $a_1$ | $\hat{a}_1$ | $a_1 - \hat{a}_1$ | $(a_1 - \hat{a}_1)$.ub |
|---|---|---|---|---|---|---|
| 5 | $*$ | $*$ | −2 | −3 | 1 | 1 |
| 10 | $*$ | $*$ | 3 | 0 | 3 | 5 |
| 5 | M | $*$ | 1 | 0 | 1 | 1 |
| 5 | F | $*$ | −3 | −3 | 0 | 0 |
| 10 | F | $*$ | 3 | 0 | 3 | 5 |

**Table 7: Cuboids Age and (Age, Sex)**

**Correctness**: We argue the correctness of the algorithm via contradiction. Suppose a cell $c$'s anomaly score $V$ is larger than the smallest value in the final top-$k$, but somehow $c$ was pruned and thus not included in the top-$k$. Suppose the pruning occurred at an ancestor cell $c_0$ whose anomaly upper bound was $V_0$. By Lemma 1, $V_0 \geq V$. There are two possible cases. First, if $V$ is larger than the smallest value in the top-$k$, $V_0$ must be as well. Thus the pruning could not have happened. Second, if $V_0$ is less than the smallest value in the top-$k$, then pruning $c$ was the correct decision, but $c$ should not be in the top-$k$. Both cases result in contradictions. ■

## 3.4 Iterative Search

After discovering the local top-$k$ cells in a subspace, they are merged into a global top-$k$. Entries from the original group are removed from the Time Anomaly Matrix. The whole process repeats until the Time Anomaly Matrix is empty. In the example of Table 4, the first iteration finds the rule: "Income = 45k–60k → Magnitude Anomaly : S[2–3]". In the next iteration, the entries with phase anomaly are selected and produce the following rule: "Education = College → Phase Anomaly : S[1]". Algorithm 2 shows a high-level summary of SUITS.

**Algorithm 2** SUITS

Input & Output: Same as Algorithm 1

```
1. Retrieve data for $\sigma_p(R)$
2. Repeat until global answer set contains global top-$k$
3.     $B \leftarrow$ candidate attribute values from $\{A_1, \ldots A_n\}$
4.     Retrieve top $k$ anomaly cells from $C_B$ using $g$ and $m$
5.     Add top $k$ cells to global answer set
6.     Remove discovered anomalies from input
7. Return top $k$ cells in global answer set
```

## 3.5 Discussion

**Top-K Approximation.** SUITS combines local top-$k$'s to form the global top-$k$. This is different from Algorithm 1 where the global top-$k$ is computed directly. Though theoretically these methods could produce different answers, we argue that it is unlikely and the benefits of SUITS outweighs the risks by far. Consider Algorithm 1 for a relative small 30-dimensional problem. It has to compute $2^{30}$ cuboids! This is simply impractical. SUITS, on the other hand, can handle it easily by partitioning the data.

Next, we examine the likelihood of SUITS producing a different answer set than Algorithm 1. Every anomaly in the data must be correlated with a subspace; in the case of top-$k$ anomalies, the correlation should be rather obvious and thus easily detectable by the AL scores. As a result, they will likely become candidate subspaces for SUITS. After the candidate subspace is chosen, the rest of the calculation (cubing and top-$k$ pruning) is lossless. The source of error is when a subspace is never chosen by SUITS. This could occur if a "bad" attribute somehow has a very low AL score; but in testing, we found this to be rare. In the next section, we show empirically that SUITS usually produces the true top-$k$. Further, it does it within a fraction of the time required by the naïve algorithm.

**Pruning.** The top-$k$ pruning mechanism relies on the satisfaction of Lemma 1. In the case of regression and our definitions of anomaly, this is possible because the anomaly values are aggregated via SUM. The anomaly values themselves might not increase or decrease monotonically, but the upper bounds do. If Lemma 1 is not satisfied (*i.e.*, non-monotonic measure), one will not be able to prune in cube computation. However, this only affects individual partitions of the algorithm. The overall divide-and-conquer search still applies. Further, because the dimensionality of each partition is relatively low, full cube computation could be tolerated.

**Optimizations.** There are many tricks one can play to speedup computation. First, if the AL score of some attribute value is infinity, it means that value is trivially correlated with the anomaly. Thus, it can be output directly without cubing. Second, in [5], dimensions are sorted in descending order of cardinality in order to maximize the chance of early pruning. The same can be applied here. By sorting dimensions in ascending order of upper bound values, the chance of pruning should be greater.

## 4. EXPERIMENTS

To show the effectiveness of SUITS, we experimented with both synthetic and real world data. SUITS was implemented in C++ and compiled with GCC. All experiments were performed on a Linux machine with an Intel Core2 E6600 CPU and 2GB of memory.

### 4.1 Real World Data

We obtained real sales data from a Fortune 500 company. For confidentiality reasons, the name of the company, the names of products, or actual sales numbers cannot be revealed. The data include records from 1999 to 2005 and contains over 925,000 sales and nearly 600 dimensions. The measure in the cube is number of sales. $g$ was computed for the entire time span and not sub-sequences.

#### *4.1.1 Sample Queries*

To perform some sample queries, we restricted ourselves to 30 of the 600 dimensions. They included age, gender, marital status, household size, occupation, employment, race, and more. Construction of 1D shell fragments around this data took approximately 8 seconds. For the first query, the probe cell was set to (Gender = "Male", Marital = "Single", Product = a luxury item). Of its descendants, (Generation = "Post-Boomer") had the largest magnitude anomaly (Figure 6); it was significantly *less* than the expected. This matches our intuitions because post-boomers are those under age 35 and probably do not have enough financial resources to purchase the luxury item. The second and third largest abnormal descendants were (Occupation = "Manager/Professional") and (Occupation = "Manager/Professional", Number of Children Under 16 = 0). Both of these segments have significantly *more* sales than the expected, and that makes sense because they have high-paying jobs and more disposable income.

For the second query, the probe cell was set to (Gender = "Female", Education = "Post-Graduate", Product = a cheap item). Of its descendants, (Employment = "Full-Time") had the largest trend anomaly; its trend was *declining* while the expected trend was relatively flat. This makes sense because presumably a full-time employee with a post-graduate degree should have relatively good financial standing such that she will purchase more expensive alternatives. The other anomalies in the top-10 had similar results (*e.g.*, Occupation = "Manager/Professional"). But the third largest anomaly in the top-10 was a little intriguing.

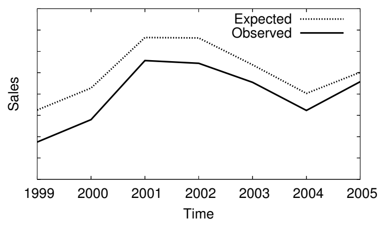


**Figure 6: Gender = "Male", Marital = "Single", Product = a luxury item, Generation = "Post-Boomer"**

With (Number of Children Under 16 = 0), the observed trend was *higher* than expected: The expected trend was 8 while the observed was 14. Figure 7 shows the data along with their least-square fit lines. One explanation could be that females who do not have children in the household tend to be relative young and thus have not had time to accumulate enough financial resources to purchase more expensive alternatives. The age dimension did not show up as anomaly here, because its values were not grouped in ranges and probably were too small individually to register in the top-$k$. We did partially verify the explanation by setting the probe to (Gender = "Female", Education = "Post-Graduate", Product = a cheap item, Number of Children Under 16 = 0). In the top-10 anomaly of this query, the *only* anomaly whose trend was unexpectedly *higher* was (Generation = "Post-Boomer" (*i.e.*, young)) (Figure 8). This partially corroborates our age conjecture.

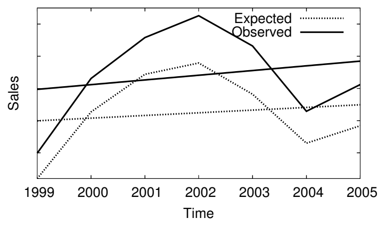


**Figure 7: Gender = "Female", Education = "Post-Graduate", Product = a cheap item, Number of Children Under 16 = 0**

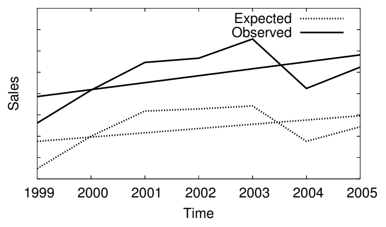


**Figure 8: Gender = "Female", Education = "Post-Graduate", Product = a cheap item, Number of Children Under 16 = 0, Generation = "Post-Boomer"**

These two sample queries show typical market analysis

| Probe | $\|R\|$ | Naïve | SUITS$_0$ | | SUITS | | Common Top-10 |
|---|---|---|---|---|---|---|---|
| | | Time | Time | % Improve | Time | % Improve | |
| Male, Single | 10 | 14 | 5.9 | 58% | 5.4 | 61% | 9 |
| Male, Married | 10 | 299 | 95 | 68% | 60 | 80% | 10 |
| Male, Divorced | 10 | 3.6 | 2.8 | 22% | 2.8 | 22% | 10 |
| Female, Single | 10 | 15 | 8.2 | 46% | 7.0 | 53% | 9 |
| Female, Married | 10 | 114 | 31.0 | 73% | 23.0 | 80% | 8 |
| Female, Divorced | 10 | 5.5 | 3.8 | 31% | 3.7 | 33% | 10 |
| Post-Boomer, Children=0 | 11 | 68.8 | 39.6 | 43% | 32.1 | 53% | 10 |
| Post-Boomer, Children=1 | 11 | 16.8 | 5.4 | 68% | 4.8 | 71% | 10 |
| Post-Boomer, Children=2 | 11 | 15.5 | 7.8 | 50% | 6.7 | 57% | 10 |
| Boomer, Children=0 | 11 | 108.9 | 75.7 | 30% | 52.4 | 52% | 10 |
| Boomer, Children=1 | 11 | 120.3 | 68.9 | 43% | 58.0 | 52% | 10 |
| Boomer, Children=2 | 11 | 46.6 | 27.2 | 42% | 23.6 | 49% | 10 |
| | | | *Average* | 48% | | 55% | 9.6 |

**Table 8: Run times of trend anomaly query with low dimensional data ($10 \leq |R| \leq 11$)**

queries and how their results can either be confirmed by intuition or shed light on some unexpected behavior. The former case offers assurance to the analyst that common-sense marketing tactics will work and the latter offers a chance to enter an un-tapped market. One may remark that the differences in trend are rather small in the examples we have shown. But the truth is, in real world market data, there are no "nice" anomalies where the trends are completely opposite. A difference in slope of just 1 or 2 is already significant.

### 4.1.2 Query Efficiency

In Table 8, we show efficiency results of many trend anomaly queries. For each query, we processed it in three different ways. First, we used the Naïve algorithm as described in Algorithm 1. Second, we used **SUITS**$_0$, which is SUITS without the top-$k$ pruning. That is, it uses the iterative subspace search but local candidate cubes are fully materialized. And lastly, we used SUITS as described in the paper. Table 9 shows a similar experiment with magnitude anomaly queries except without SUITS$_0$. In both tables, $|R|$ shows the total number of dimensions in the data; we chose a relatively small number because for even slightly larger datasets, Naïve is orders of magnitudes slower.

| Probe | $\|R\|$ | Naïve | SUITS | | Common |
|---|---|---|---|---|---|
| Male, Single | 10 | 13.4 | 8.2 | 38% | 10 |
| Male, Married | 10 | 182.4 | 46.9 | 74% | 10 |
| Male, Divorced | 10 | 4.1 | 3.1 | 24% | 10 |
| Female, Single | 10 | 15.4 | 7.7 | 50% | 10 |
| Female, Married | 10 | 85.5 | 17.4 | 80% | 9 |
| Female, Divorced | 10 | 6.5 | 4.1 | 37% | 10 |
| High School | 11 | 92.5 | 22.9 | 75% | 10 |
| College | 11 | 382.4 | 35.5 | 91% | 10 |
| Post-Graduate | 11 | 110.7 | 34.9 | 68% | 10 |
| | | | *Average* | 60% | 9.9 |

**Table 9: Magnitude anomaly query run times**

In both tables, we notice that SUITS is, on average, over 50% faster than the Naïve algorithm. This is especially true for the large queries, either by a large number of dimensions or a large number of tuples. This is not surprising because SUITS breaks a very large problem into more manageable parts. Figure 9 shows a closer look at a single query as the number of dimensions increase from 7 to 14. At 7, the Naïve algorithm is faster than SUITS, because the data cube is relatively small and SUITS has additional overhead. However, as dimensionality increases, the trends of Naïve and SUITS are very different. Naïve shows the expected curse of dimensionality; in fact, with $|R| = 12$, Naïve ran out of memory for full cube materialization. With SUITS, we observe a more or less linear or even sub-linear behavior. This is because SUITS is more dictated by the anomalies inside the data rather than the external size of the data.

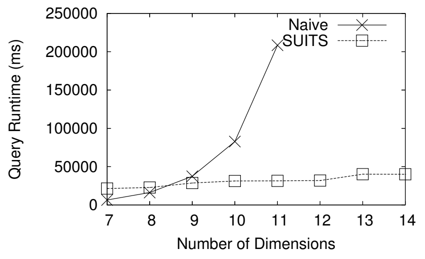


**Figure 9: Running time vs. number of dimensions**

As mentioned previously, the top-$k$ produced by SUITS is not guaranteed to be the same as the true top-$k$. This could occur if particular attributes or combinations of attributes are not examined within a single iteration of SUITS. In practice, we noticed that this sometimes happens with dimensions of high cardinality (*e.g.*, zip code, state). The reason is that high-cardinality dimensions often have high entropy just by definition and thus low AL scores. And so they sometimes are not picked as candidates. An easy way to fix this would be to normalize entropy based on the cardinality of the dimension. However, this scenario is usually the exception rather than the rule. The last columns of Tables 8 and 9 show the number of items in the top-10 that is common between the SUITS top-10 and the true top-10. As shown, SUITS usually produces the same top-10 as the true top-10.

## 4.2 Synthetic Data

To test SUITS in a more controlled environment, we also generated our own data. Each data set consisted of 95% normal, background "noise" and 5% abnormal patterns. For the normal portion, each value under each dimension was picked uniformly and independently between 1 and 5. Each value inside the count and time series measure was also picked uniformly and independently between 0 and 5. For the abnormal portion, 10 abnormal patterns were generated. Each pattern consists of 1–3 dimensions and a randomly chosen count/measure pattern. These 10 patterns are randomly inserted into the data 5% of the time. For all queries in the rest of this section, $p$ was set to a random 2-dimension query. Each experiment was repeated 10 times to get an average.

Figure 10 shows running times in a 10-D data set as the number of tuples increases with everything else fixed. Both SUITS and Naïve exhibit linear complexity; even though SUITS is iterative. This is expected because the size of the Time Anomaly Matrix grows linearly with data size and so does the number of iterations needed to cover it.

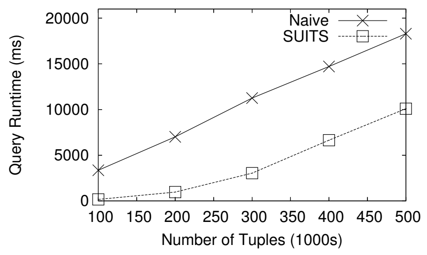


**Figure 10: Running time vs. number of tuples**

Curse of dimensionality is a well-known problem in data cubing and other multi-dimensional analysis. In SUITS, because bulk of the analysis is spent on subspaces, the overall dimensionality should not affect running time too much. Figure 11 shows running time as dimensionality increases from 6 to 16 but everything else remained the same. The number of tuples was 250,000. With the Naïve method, the running time quickly runs out of control. With SUITS however, we observe a relatively linear or flat curve. The flat portion is explained by the sparsity of the high dimensional data. In these cases, SUITS may only run a couple of iterations because anomalies are so easy to find.

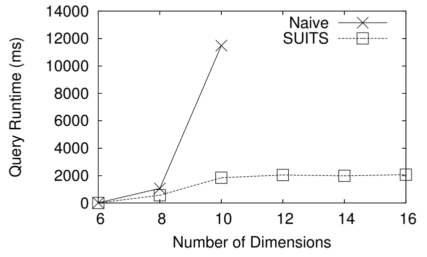


**Figure 11: Running time vs. number of dimensions**

One of the reasons we chose regression as the method of representation for time series data was that it allows aggregation from intermediate results as opposed to from scratch. Figure 12 corroborates this assertion. The data had 8 dimensions and 250,000 tuples. As the length of the time series increases from 20 to 100 with everything else fixed, one observes a linear increase for the Naïve method. This is expected because there are just more numbers to aggregate. For SUITS, the increase is much flatter since the length of time series does not affect aggregation of regression parameters. The minor increase is probably due to the increase in time to compute the least-square fit.

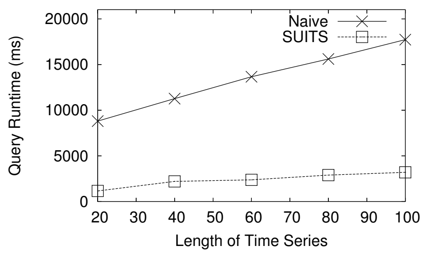


**Figure 12: Running time vs. length of time series**

## 5. RELATED WORK

Anomaly detection in data cubes or Exploratory Data Analysis [11] in not a new problem. [24] developed a model of OLAP discovery guided by anomalies. Their definition of anomaly is measured on an anticipated value, which is computed from *all* related group-by's in the cube. This necessitates full materialization of the data cube, which is prohibitive in high dimensional cases. Further, the mode of computation is interactive, whereas our work is automated search. [4] uses the median polish to fit additive models in a data cube. This facilitates approximate exploratory analysis and also reduces I/O in the process; however, it does not address the high-dimensionality problem. [17] is another work which provides approximate answers to queries and finds interesting cells in a data cube. Principles from maximum entropy are used to compress the data; however, calculating the marginals still has exponential complexity with respect to the number of dimensions.

[12] and [9] consider the cubegrade problem in association patterns and also data cubes. A cube cell is marked as interesting if the ratio between its measure and the measure of another cell exceeds some threshold. A major restriction of these proposals is that the measures used have to satisfy certain anti-monotonic property, which enables pruning. Our work does not make this assumption. Lastly, in all these works including [24], the data cube measures are scalars while SUITS focuses on data with time series.

[8] from the time series field also addresses a similar problem, but there are two major differences. First, SUITS's search is automated while [8] requires user-guidance. Second, SUITS works for high-dimensional data while [8] is only suitable for low-dimensional data ($< 10$). In [8], the user is expected to define the observation/minimum layers and popular paths. After these restrictions and pre-computations, the leftover space is assumed to be small enough to be computed by a traditional cubing method (*e.g.*, H-cubing in [8]). If the leftover space still has more than 10 dimensions, H-cubing will have a difficult time computing the exceptions. In SUITS, no domain knowledge is required, and the original high-dimensional cuboid space is automatically partitioned (both along dimension lines and tuple lines) into more manageable parts.

General OLAP research has produced many data cubing algorithms [5, 15, 16]. More advanced cubing techniques can be incorporated into SUITS to enhance the efficiency, but the overall goal of SUITS is different from cube materialization or summarization.

Outlier detection in multi-dimensional data contains much work [21, 18, 22]. [21] clusters the data into partitions and processes them separately much like SUITS. The idea is that the $k^{th}$ nearest neighbors lie close and would be in the same partition. However, SUITS takes this one step further by examining the subspaces of the partitions. Also of interest are high-dimensional cases. [2] uses an evolutionary algorithm to avoid the curse of dimensionality. [7] uses the $K$-d tree to store points, which reduces complexity to linear with regards to dimensionality. However, the data model in general outlier detection is completely different. Between any two points, it measures anomaly on the distance between their dimensional values (*e.g.*, Euclidean). In the OLAP model, there is no such distance between cube cells. Anomaly is defined on the time-series measures associated with the points. Also, aggregation and expectation introduce new challenges that require new solutions.

Subspace clustering [19] detects clusters in subspaces of a high-dimensional space via either top-down [1] or bottom-up [3] exploration. SUITS contains a similar spirit though the method by which subspaces are selected is quite different. Instead of adding or removing dimensions at each iteration, a completely different set is chosen that is based on the nature of the data. But more importantly, subspace clustering, like outlier detection, measures anomaly on the points, not the OLAP measures, aggregates, and expectations.

SUITS uses piecewise linear regression to represent time series and compute anomalies. Work in time series, specifically similarity search, addresses similar issues. Robust solutions [20, 13, 23, 25, 10] are able to find similar shapes invariant to many transformations. [14] finds the most unusual sub-sequence within a set. Work in data streams [27, 6] also provides tools for finding surprises in time series-like data. We believe these algorithms are orthogonal to ours in that we are more focused on the OLAP aspects. A typical time-series problem is conducted over a flat set (or predefined aggregate functions) while ours is over a structured high-dimensional space that can be arbitrarily aggregated.

## 6. CONCLUSIONS AND FUTURE WORK

Anomaly detection in multi-dimensional time series is clearly an important problem with many applications. Market analysis is just one of many. While much work has been devoted to OLAP and time series, little attention has been paid to the combination of the two. In real world applications, the combination is often exactly where the useful answers lie. In this paper, a solution, SUITS, is proposed to find top-$k$ anomalies within a high dimensional time series data cube. SUITS avoids the curse of dimensionality by examining only the most promising subspaces in an iterative manner. In experiments with real world sales data, this was shown to be both effective and efficient.

In our future work, we plan to address two issues. First, semantic compression and closed computations [26] could be useful in the real world because many patterns differ only slightly and offer little incremental information gain. Numerous techniques have been proposed and it would be interesting to apply them here. Second, in Section 3.5, it was mentioned that other time-series similarity measures maybe adopted if they satisfy some properties. In this paper, the simplest anomaly measures were used, which, by no means, represent the best time-series research has to offer. It would be very useful to incorporate some of the more powerful similarity measures into SUITS.

## 7. REFERENCES

[1] C. C. Aggarwal and P. S. Yu. Finding generalized projected clusters in high dimensional spaces. In SIGMOD'00.

[2] C. C. Aggarwal and P. S. Yu. Outlier detection for high dimensional data. In SIGMOD'01.

[3] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In SIGMOD'98.

[4] D. Barbará and X. Wu. Using approximations to scale exploratory data analysis in data cubes. In KDD'99.

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

[6] A. Bulut and A.K. Singh. A unified framework for monitoring data streams in real time. In ICDE'05.

[7] A. Chaudhary, A. S. Szalay, and A. W. Moore. Very fast outlier detection in large multidimensional data sets. In DMKD'02.

[8] Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang. Multi-dimensional regression analysis of time-series data streams. In VLDB'02.

[9] G. Dong, J. Han, J. Lam, J. Pei, K. Wang, and W. Zou. Mining constrained gradients in multi-dimensional databases. In TKDE'04.

[10] A. W. Fu, Eamonn Keogh, Leo Yung Hang Lau, and Chotirat Ann Ratanamahatana. Scaling and time warping in time series querying. In VLDB'05.

[11] D. C. Hoaglin, F. Mosteller, and J. W. Tukey. *Understanding Robust and Explorartory Data Analysis.* Wiley, 1986.

[12] T. Imielinski, L. Khachiyan, and A. Abdulghani. Cubegrades: Generalizing association rules. In DMKD'02.

[13] E. Keogh and S. Kasetty. On the need for time series data mining benchmarks: A survey and empirical demonstration. In KDD'02.

[14] E. Keogh, J. Lin, and A. Fu. Hot sax: Efficiently finding the most unusual time series subsequence. In ICDM'05.

[15] X. Li, J. Han, and H. Gonzalez. High-dimensional OLAP: A minimal cubing approach. In VLDB'04.

[16] K. Morfonios and Y. E. Ioannidis. Cure for cubes: Cubing using a rolap engine. In VLDB'06.

[17] T. Palpanas and N. Koudas. Entropy based approximate querying and exploration of data cubes. In SSDBM'01.

[18] S. Papadimitriou, H. Kitagawa, P. B. Gibbons, and C. Faloutsos. LOCI: Fast outlier detection using the local correlation integral. In ICDE'03.

[19] L. Parsons, E. Haque, and H. Liu. Subspace clustering for high dimensional data: A review. *SIGKDD Explorations*, 6:90–105, 2004.

[20] I. Popivanov and R. J. Miller. Similarity search over time-series data using wavelets. In ICDE'02.

[21] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In SIGMOD'00.

[22] P. J. Rousseeuw and A. M. Leroy. *Robust Regression and Outlier Detection.* John Wiley & Sons, Inc., 2002.

[23] Y. Sakurai, M. Yoshikawa, and C. Faloutsos. Ftw: fast similarity search under the time warping distance. In PODS'05. 2003.

[24] S. Sarawagi and G. Sathe. Intelligent, interactive investigation of OLAP data cubes. In SIGMOD'00.

[25] H. Wu, B. Salzberg, G. C. Sharp, S. B. Jiang, H. Shirato, and D. R. Kaeli. Subsequence matching on structured time series data. In SIGMOD'05.

[26] D. Xin, J. Han, X. Yan, and H. Cheng. Mining compressed frequent-pattern sets. In VLDB'05.

[27] Y. Zhu and D. Shasha. Efficient elastic burst detection in data streams. In SIGMOD'03.