Hierarchical Summaries of Change in Multidimensional Data

by

Alexandra Kim

B.Sc., Nazarbayev University, 2016

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate and Postdoctoral Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

January 2019

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, a thesis entitled:

Hierarchical Summaries of Change in Multidimensional Data

submitted by **Alexandra Kim** in partial fulfillment of the requirements for the degree of **Master of Science** in **Computer Science**.

Examining Committee:

Laks V.S. Lakshmanan, Computer Science
Supervisor

Ed Knorr, Computer Science
Supervisory Committee Member

Abstract

Multidimensional data is prevalent in data warehousing and OLAP. Changes to the data in a data warehouse are of particular interest to analysts as well as knowledge workers since they may be instrumental in understanding trends in a business or enterprise. At the same time, an explicit enumeration of all changes to the fact tuples in a data warehouse is too verbose to be useful; instead, a summary of the changes is more desirable since it allows the user to quickly understand the trends and patterns. In this thesis, we study the problem of summarizing changes in hierarchical multidimensional data with non-overlapping but containable hyperrectangles. An advantage of such summaries is that they naturally follow a *tree structure* and therefore are arguably easy to interpret. Hierarchies are naturally present in data warehouses and our constructed summaries are designed to leverage the hierarchical structure along each dimension to provide concise summaries of the changes.

We study the problem of generating lossless as well as lossy summaries of changes. While lossless summaries allow the exact changes to be recovered, lossy summaries trade accuracy of reconstruction for conciseness. In a lossy summary, the maximum amount of lossiness per tuple can be regulated with a single parameter α. We provide a detailed analysis of the algorithm, then we empirically evaluate its performance, compare it to existing alternative methods under various settings and demonstrate with a detailed set of experiments on real and synthetic data that our algorithm outperforms the baselines in terms of conciseness of summaries or accuracy of reconstruction w.r.t. the size, dimensionality and level of correlation in data.

Lay Summary

It is common for data warehouses to store multidimensional data that is naturally hierarchical (e.g., geographic locations, store categories, etc.). Data analysts exploring the data oftentimes try to find patterns of change between two time snapshots, such as sales increase/decrease in the current month compared to the previous month. This dissertation presents a method of building tree-structured summaries of change data to facilitate exploration and pattern discovery by the analysts and knowledge workers.

Preface

This thesis is submitted in partial fulfillment of the requirements for a Master of Science Degree in Computer Science at the University of British Columbia. All work presented in this dissertation is original work of the author, performed under the supervision of Prof. Laks V.S. Lakshmanan. The author of the dissertation was the main author of this work and was also responsible for the implementation and analysis of this work. Dr. Divesh Srivastava, AT&T Labs-Research, and Prof. Lakshmanan were involved in the project idea development, discussions and contributed to manuscript edits. Prof. Ed Knorr provided feedback on the final write-up of the dissertation.

Table of Contents

List of Figures

Acknowledgements

This work would not have been possible without the patient and careful guidance of my supervisor Prof. Laks V.S. Lakshmanan and our collaborator, Dr. Divesh Srivastava, both of whom have helped me greatly during the course of this work, both educationally and morally, providing feedback and support whenever I needed it.

I would like to thank Prof. Ed Knorr for providing feedback on the writing of this thesis; his attentiveness to detail helped greatly in making this dissertation more readable and consistent.

I am also grateful to the Data Management and Mining Lab and all of its members for insightful discussions, educational reading groups and memorable moments that we have had during my journey at UBC. To all graduate students at Tuesday Tea and UDLS, thank you for the meaningful and fun conversations that helped keep me sane and optimistic during the course of my program.

Last but not least, I would like to thank my boyfriend Amon and my family for all their love and support.

Chapter 1

Introduction

Common tasks of a data analyst include finding trends and patterns in data [11]. This includes analyzing the changes in data between two time points, such as comparing sales quantities of every product between two months at a network of stores. The simplest approach is to enumerate all changes (i.e., for every product and every store location), however such enumeration quickly becomes too verbose and difficult to work with. A more concise summary of change is desired in such scenarios.

A natural idea is to *summarize* the changes in the data. Given that a data warehouse is equipped with hierarchies over each of its dimensions, we could leverage these hierarchies for the purpose of constructing the summaries. Previous works have attempted to summarize change with overlapping or disjoint hyperrectangles and have faced hardness issues. We present summaries consisting of non-overlapping but containable hyperrectangles that concisely reflect change between the snapshots. This allows to construct optimal-size summaries in polynomial time that are also conceivably easier to interpret due to their natural tree-like structure.

Consider the following scenario.

1.1 Example

A data analyst at an outdoor equipment store is interested in seeing what product types have undergone the most percentage increase/decrease in demand in the past month. The analyst takes the ratio between new and old sales across products and locations (see Figure 1.1).

We may see that the sales have undergone some moderate fluctuations. Notably, across all locations, the store has sold more insect repellent products compared to the previous month. The sales of camping equipment have slightly gone up at the Manhattan store and moderately decreased for the Queens store. Additionally, there is a slight dip in sales of sleeping bags at the Newark store and a similar decrease in sales of first aid kits at the Manhattan branch. While manual inspection works well for small amounts of data, it is not practical to draw such conclusions from mere inspection in

the case a of large multidimensional data warehouse. In fact, even a detailed listing of only {location, product} combinations where there was a considerable (upward or downward) change in demand can consist of thousands (if not many more) of tuples and can be hard to assimilate for an analyst or a knowledge worker.

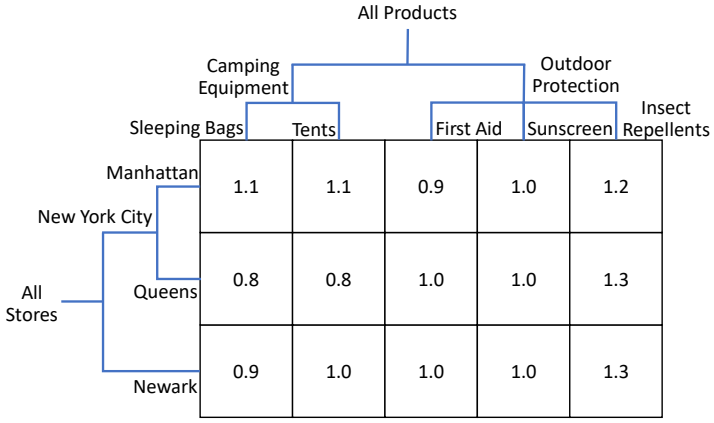

Figure 1.1: Example. Change data as a ratio between new and old sales at an outdoor equipment store.

If we enumerate all tuples where any change has occurred (i.e., the change ratio is not 1) the size of such a summary is 9. However, by using hierarchical structure and allowing containment of the regions we can achieve significantly smaller summaries. Such a summary is presented in Figure 1.2. The summary contains 7 rectangles and is thus of size 7. This summary is lossless, meaning that it is possible to reconstruct all original tuples *exactly* by looking up the smallest rectangle containing the tuple.

So, we build summaries that exhibit the following semantics: for every data tuple, its value is equal to the weight of the smallest ancestor rectangle in which it is contained. This way, we leverage the hierarchical nature of the data to maintain a smaller summary. Rectangles in this case are *hierarchy-constrained* meaning that every rectangle corresponds to a roll-up of data cube cells that the rectangle covers. Calculation of change values is orthogonal to this approach and is mostly application based: an analyst could decide to take a ratio or a difference between values in two time snapshots, calculate the percentage change, or express changes in any other sensible way. For the rest of this paper, we use ratios as our base change data. In our discussions, we also assume that rectangles are hierarchy-constrained, whenever hierarchies are present.

Additionally, if we allow some minor loss of information we could get a further decrease in summary size. If the data analyst is willing to tolerate 5% reconstruction error per tuple, meaning that the reconstructed value is no more than 5% off from the original value, we reduce the summary size from 7 to 5, as shown in Figure 1.3. The amount of error can be easily regulated by tweaking the parameter α. For instance, if we allow a 10% error, the summary size further decreases to 4. Such lossy summaries help to achieve a significantly more compact size and may be favored by the analysts.

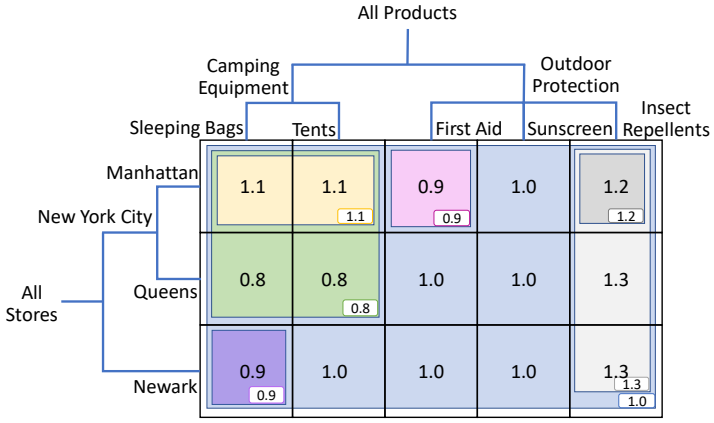

Figure 1.2: Summary of the outdoor equipment store sales data changes. The weight of each rectangle is shown in its bottom right corner.

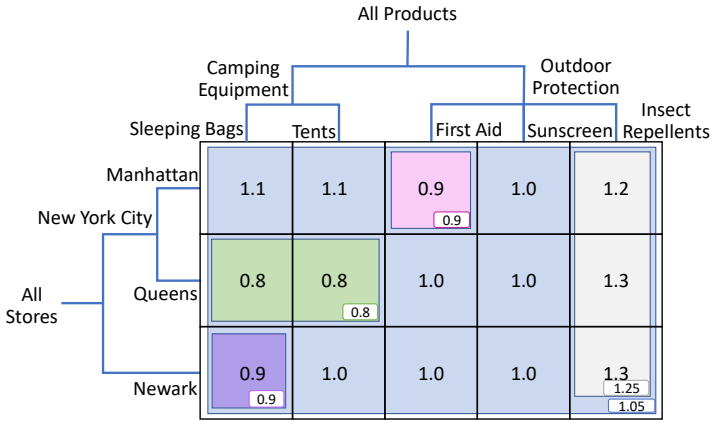

Figure 1.3: Summary of the outdoor equipment store sales data with max loss of 5%.

1.2 Alternative Summary Methods

While there has been substantial work on compressing and summarizing multidimensional data cubes, there are two key papers which share our goal and we briefly describe them next and compare our approach with theirs in later sections.

Karloff et al. study a similar summarization problem in [8]. Their summaries consist of hierarchy-constrained overlapping rectangles, and each tuple's original value is reconstructed by summing up weights of all rectangles in which the tuple is contained. Although such formulation is versatile due to allowed overlaps it leads to intractability: they show that finding optimal summaries is NP-hard. However, they do propose an efficient randomized 2-approximation algorithm for this problem. Notably, they do not consider $d > 2$ dimensions and it is unclear if their approach can be extended to provide a constant factor approximation for d dimensions. In addition, they only consider lossless summaries and it is an open problem whether it is possible to use their framework to trade accuracy for summary size in a principled manner, while providing a guaranteed approximation. The summary generated by Karloff et al.'s algorithm is shown in Figure 1.4. The algorithm was run 100 times, and the summary size varied from 8 to 16 across runs.

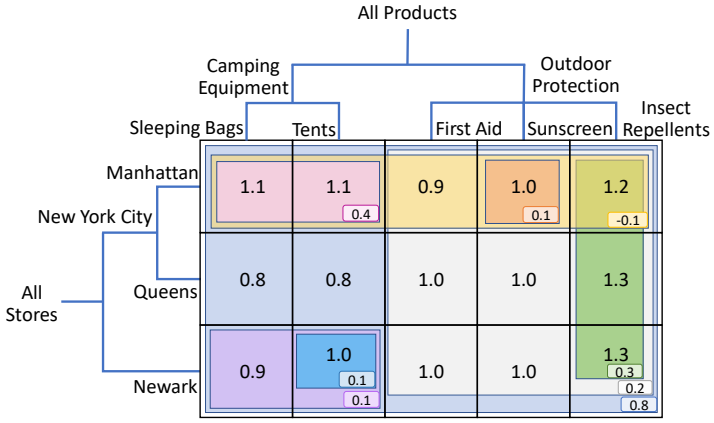

Figure 1.4: Lossless summary of the equipment store sales by Karloff et al. [8]. The size of the summary is 8.

Ruhl et al. [13] study the problem of summarizing changes in multidimensional data from a different perspective. Given a desired budget on the size of the summary, their algorithm finds a summary whose size is under

budget. Their summaries are inherently lossy and are not designed for reconstruction of original tuple values. The summaries correspond to sets of non-overlapping hierarchy-constrained rectangles which are responsible for the maximum change. The summary generated using Ruhl et al.'s approach is shown in Figure 1.5.

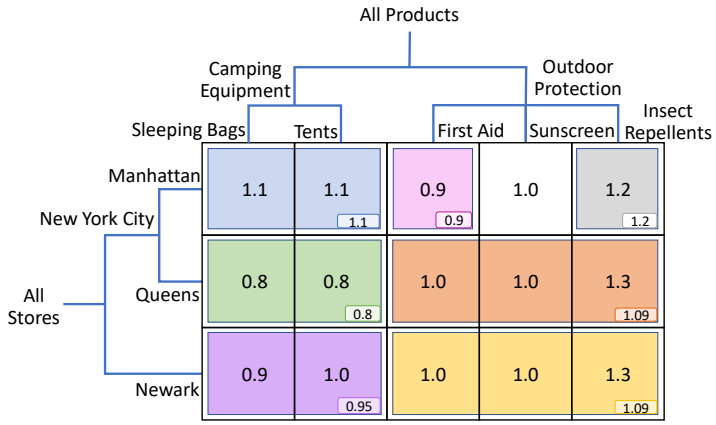

Figure 1.5: Lossy summary of the equipment store sales by Ruhl et al. [13]. The size of the summary is 7. The rectangle weights (in bottom right corners) are the reconstructed average values. More on reconstruction in Section 6.

1.3 Contributions

The contributions of this paper are as follows:

- We formulate the problem of obtaining summaries of multidimensional data having dimension hierarchies.
- While previous work [8, 13] has faced hardness results, our formulation allows us to develop an efficient polynomial-time optimal algorithm.
- Our algorithm naturally transitions from lossless to lossy summaries with the ability to bound the worst case reconstruction error.
- We show empirically that summaries produced by our approach are more concise than those by [8] that provide a randomized 2-approximation guarantee, and have a smaller reconstruction error than [13] for summaries of an equivalent size.

Chapter 2

Related Work

2.1 Lossless Reconstruction Summaries

Agarwal et al. study the problem of summarizing single-dimensional data with tree structures [1]. Their goal is to assign weights to hierarchy-constrained intervals, so that for every tuple, its original value is equal to the sum of weights of all intervals it is a descendant of. They propose an efficient polynomial time algorithm; however, the main constraint is that the algorithm can be applied for data containing only one dimension.

Karloff et al. extend Agarwal et al.'s one-dimensional solution to two dimensions [8]. Their summaries consist of hierarchy-constrained overlapping rectangles and while this class of summaries can be versatile due to allowed overlap, it runs into hardness issues even in 2 dimensions. As of right now, the algorithm can operate only in 2 dimensions. It provides a randomized 2-approximation to build lossless summaries.

Bu et al. [2] presented an MDL (minimum description length) problem in which there are "interesting" tuples, as well as tuples of no interest. A description is a set of hierarchy-constrained overlapping rectangles that are required to cover all interesting tuples and any number of uninteresting tuples, however every found rectangle is counted towards the length of the description. Mendelzon and Pu explore the MDL problem for various structures [12]; of a particular interest to this work is MDL with hierarchies in which a hierarchical structure is utilized to reduce the description length whenever possible. They also study multidimensional hierarchies and introduce three languages for describing subsets of the universe, all of which were shown to be NP-complete even in a 2-dimensional case.

2.2 Lossy Reconstruction Summaries

The approach by Agarwal et al. discussed in 2.1 finds optimal weighting schemes for a one-dimensional version of our problem and allows finding both lossless and lossy summaries by regulating a per-node worst case re-

construction error with a tolerance parameter ϵ [1].

El Gebaly et al. tackled a related summarization problem [3]. They have developed the concept of explanation tables as a way to present concise summaries of multidimensional relations with a binary outcome attribute. To measure the *goodness* of the summary they use Kullback-Leibler (KL) divergence between the true distribution of values and the maximum-entropy estimate implied by the table. The goal is then to find a minimum number of overlapping rectangles that satisfy a KL-divergence threshold τ, where each rectangle is assigned the count and average of the values it covers. A related problem on rule mining on a numeric measure attribute has been addressed in [4]. They develop an efficient distributed rule mining framework for tall and wide tables that outputs a desired number of rules that are found by minimizing KL-divergence, similar to [3].

Lakshmanan et al. presented a work on summarizing regions of interest in both numerical and hierarchical domains [9]. In that framework, each data cell is marked as blue, red or white (semantically corresponding to "include", "exclude" and "don't care"). The goal is to find the smallest set of rectangles, covering all blue cells, none of the red cells, and the summary may include white cells up to a certain budget. Although it is assumed that cells in the summary are blue, some of them may in fact be white; that introduces some notion of "accuracy" of the summary, and a zero budget effectively corresponds to finding a lossless summary of the regions.

Jagadish et al. [6] study compression and pattern discovery by grouping relational records into so-called fascicles, such that the range of each of the k attributes, does not exceed some distance as defined by parameter t. In pattern discovery, the goal is to find overlapping rectangles that cover at least m tuples such that the quality of fascicles is maximized (smaller t and larger k correspond to higher quality fascicles). Although reconstruction is not explicitly addressed in this work, the amount of worst-case reconstruction loss could be regulated with t.

Sarawagi introduces an advanced exploration operator for data cubes designed to explain differences in data [14]. They introduce an information theory formulation for reconstruction error and trade off the summary size and the loss in accuracy. This approach was designed to automate a manual search for change explanation. Here, instead a user is presented with a fixed (but configurable) number of rows that best explain the change.

2.3 Identifying Extreme Aggregates

A considerable amount of research has also been done on summarizing aggregate values. The main difference from our problem is that those works usually focus on aggregates directly and do not aim to reconstruct the original cell level data.

In their recent work, Ruhl et al. presented the Cascading Analysts (CA) algorithm, designed for building concise summaries of hierarchical data [13]. Every tuple corresponds to a unique hyperrectangle and is assigned a weight which is equal to the absolute sum of values of all tuples that it covers. A larger weight corresponds to a larger change in the data. Given an input k, the goal of CA is to find a bounded number (of up to k) of non-overlapping rectangles, so that the sum of weights of the rectangles is maximized. Their algorithm was shown to find an optimal solution in 2 dimensions and has an approximation ratio of $\log^{d-2}(n+1)$ for 3 and more dimensions, where n is the total number of nodes in a lattice and d is the number of dimensions. This approach is lossy, and although it may provide lossless summaries, these scenarios are very rare, e.g., all tuples undergo the same amount of change.

Wen et al. explore ways to compactly represent query results [22]. They designed a framework that allows interactive exploration of query results by finding k non-overlapping rectangles (clusters) with maximum average values covering top L tuples from the query output. Every rectangle is required to be at least at a distance D from other rectangles, where D is defined as a number of attributes that do not have the same value in the domain.

Joglekar et al. study exploration and summaries of relational tables [7]. They group tuples by their aggregate attributes. Each group is represented by a rule that describes the instantiated attributes and the aggregate value. The goal is to find a set of rules of a maximum total score, which depends on marginal coverage and specificity of the rule. The results are presented in a table for a user to explore.

2.4 Summarizing All Aggregates

Vitter et al. [20, 21] present an approach for efficient multidimensional aggregate computation based on the notion of wavelets; wavelet coefficients are precomputed and stored. Queries are then answered approximately by choosing the k most important wavelet coefficients. The amount of information loss can be regulated by the parameter k.

Related to the problem of data aggregate summarization lies the work by Lakshmanan et al. [10] that presents a structure that they call a quotient cube. Given an aggregate function, a quotient cube finds a reduced cube lattice of the aggregate values by partitioning the lattice cells into clusters.

2.5 Other

Sismanis et al. [18] presented a dwarf cube – a data cube compressed by eliminating prefix and suffix redundancies. That data structure stores data with 100% precision and can be used to store a full cube or, alternatively, precomputed aggregates whose computation is too costly to be done on the fly. A dwarf cube can represent the actual data; however, does not aim to provide a summary.

Exploration of data cubes has been addressed by Sarawagi et al. in [16] where they present a tool for effective exploration and navigation of a data cube which is also designed for finding exceptional values. First, aggregate values of each group-by are computed bottom up. Then the algorithm goes root-down finding the difference between expected and actual values at each group-by. The group-by's with largest differences are then highlighted in a UI to signal significance to the user. In [15, 17], in addition to the data cube operator introduced in [14], Sarawagi and Sathe present two more navigational operators that automate the exploration process for a user, helping her to see a bigger picture of the data changes using a maximum entropy approach that finds interesting unvisited parts of the data cube.

Chapter 3

Problem Definition

In our work, we consider multidimensional datasets. This setting is very common for data warehouses and data analysis applications. Every dimension of such data is typically organized in the form of a hierarchy; such hierarchies may be modeled as trees or DAGs. In this work, we consider tree hierarchies. For example, the *Location* dimension may be modeled using the *Country* $\rightarrow$ *Region* $\rightarrow$ *Division* $\rightarrow$ *State* hierarchy. Other dimensions such as *Product*, *Demographics*, *Occupation*, etc. may have their own hierarchical structure. As a concrete example, consider a data warehouse over the dimensions *Product* and *Location*, as in Figure 1.1[1]. Having multiple hierarchies naturally leads to a product space, i.e., a lattice defined by the product of the dimension hierarchies. Each node of the lattice corresponds to a vector of nodes of the component dimension hierarchies.

3.1 Preliminaries

Denote the number of dimensions by d and let L be the lattice structure corresponding to the cross product of individual dimension hierarchies T^i, i.e., $L = T^1 \times T^2 \times ... \times T^d$. Let $\vec{p} = (p^1, ..., p^d)$ be a lattice node in L, where p^i is a corresponding tree node in T^i. A lattice node $\vec{p}$ is said to be a parent of a node $\vec{q} = (q^1, ..., q^d)$, if there is $j \in [d]$ such that q^j is a child of p^j in hierarchy T^j and $\forall i \neq j$, $p^i = q^i$.

A lattice node $\vec{p} = (p^1, ..., p^d)$ for which $\forall i \in [d]$, p^i is a leaf in its corresponding dimension hierarchy T^i, is said to be a lattice leaf, also referred to as a *cell*.

If two lattice nodes $\vec{q}$ and $\vec{r}$ share a common parent $\vec{p}$, they are referred to as siblings. Two siblings are called *direct* siblings, iff they share a common parent and there is some j, s.t. $r^j \neq q^j$ and $\forall i, i \neq j, r^i = q^i$. For example, the nodes *"Camping Equipment, All Stores"*, *"Outdoor Protection, All Stores"*, *"All Products, NYC"* and *"All Products, Newark"* in our previous example are siblings since they all share the common parent *"All Products,*

[1]Real data warehouses may have many more dimensions.

All Stores". Among those, *"Camping Equipment, All Stores"* and *"Outdoor Protection, All Stores"*, as well as the pair of *"All Products, NYC"* and *"All Products, Newark"*, are the examples of direct siblings.

For each internal lattice node $\vec{p}$, we partition $\vec{p}$'s children into maximal subsets of direct siblings. For convenience, we modify the classic lattice structure by introducing a grouping node Dim_j for every maximal set of direct siblings which correspond to a given dimension j, $j \in [d]$. These grouping nodes are assigned as parents of the subset that they are associated with and are in turn the children of the original parent node $\vec{p}$. We will refer to these grouping nodes as *dimension* nodes.

Figure 3.1 depicts an (incomplete) structure of the lattice in the equipment store example. The lattice nodes *"Camping Equipment, All Stores"* and *"Outdoor Protection, All Stores"* are examples of direct siblings grouped under the dimension node Dim_P of their parent *"All Products, All Stores"*, which happens to be the root of the lattice. Dim_P and Dim_L are the dimension nodes and the letters P and L stand for product and location dimensions, respectively. Given a lattice node, it may be drilled down along any dimension. The use of dimension nodes for grouping purposes makes this drill-down explicit.

We use the modified lattice structure to define the notion of *embedded trees*. An embedded tree is obtained by starting at the root, marking it, and then alternating the following operations until no longer possible, i.e., leaves are reached:

- For each marked internal lattice node, choose exactly one dimension child and mark it.
- For each marked dimension node, choose all its (lattice) children and mark them.

The structure induced by all marked nodes by the above process is an *embedded tree*. We make use of embedded trees for constructing concise summaries of multidimensional data. Note that in Figure 3.1, if we ignore grayed out regions and dimension nodes, we obtain one of the embedded trees of the lattice.

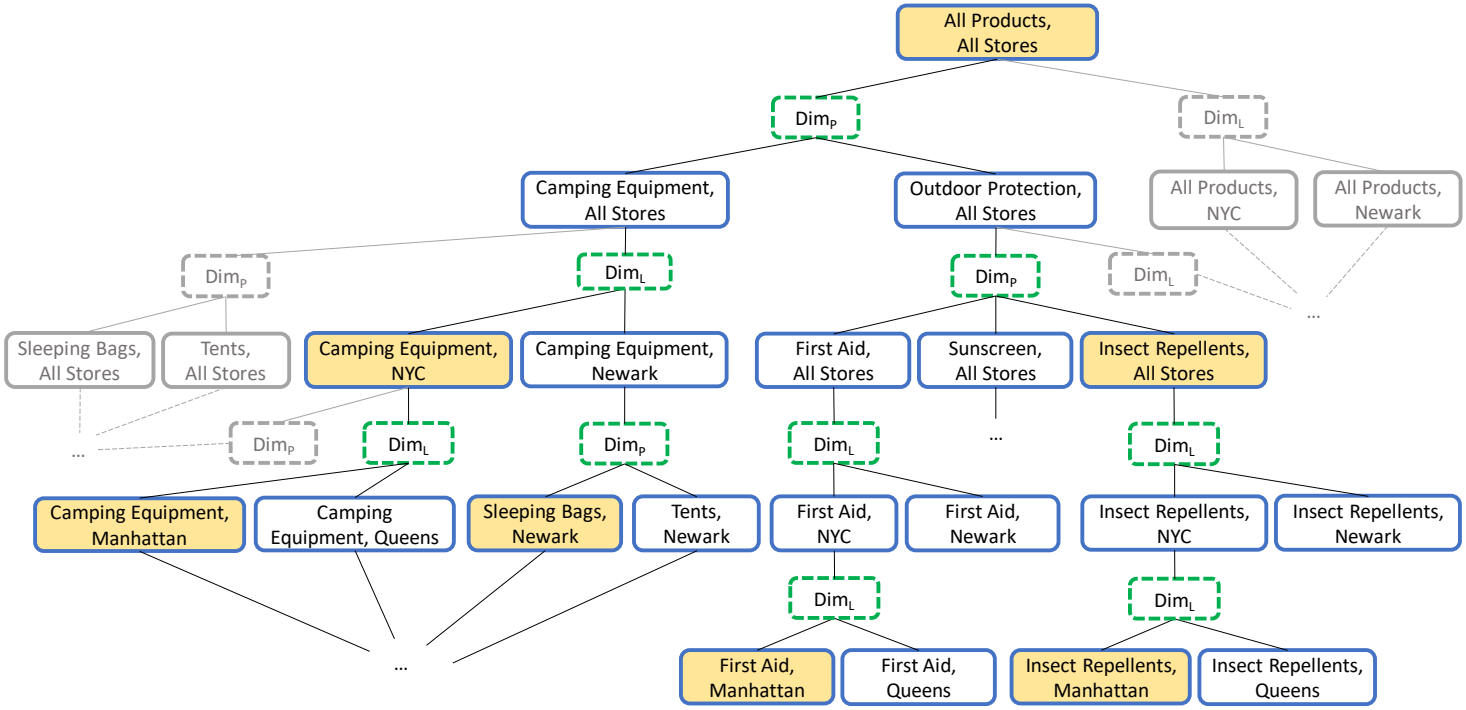

Figure 3.1: Lattice in the equipment store example. Grayed out regions are parts of the lattice outside of the (not grayed out) embedded tree. Solid blue and dashed green borders correspond to regular and dimension nodes respectively. Nodes with yellow background are the weighted nodes in the embedded summary tree from Figure 1.2.

3.2 Problem Statement

Consider a multidimensional lattice with each cell associated with some data value. This lattice could directly correspond to a data warehouse or it could be the result of operations on a data warehouse, such as comparing two snapshots and computing their ratios, differences, etc. Our goal is to find a concise tree summary of such a lattice.

3.2.1 Lossless Case

We define a *tree summary* to be an embedded tree T of the lattice with weights assigned to a selected subset of nodes such that:

- T spans all leaves of the lattice;
- the value of each lattice cell (i.e., leaf) is equal to the weight of the closest weighted ancestor of the leaf in T.

Any such summary tree is said to be a valid summary of the lattice. Clearly, there exist multiple such trees. Our goal is to find the most "concise" tree summary. For purposes of defining the size of a summary, we associate a unit cost with each weighted node of the summary tree. In other words, the cost of a summary tree T is defined as:

$$cost(T) = \sum_{t \in T} [w_t \neq \varnothing],$$

where w_t is the weight of a summary tree node t and $[w_t \neq \varnothing]$ is an indicator function that is 1 iff node t has a weight w_t assigned to it. Intuitively, the size (cost) of a summary tree is the number of nodes which have an associated weight. The goal is to find a valid embedded summary tree, so that the cost of the tree is minimized.

The basic problem statement as described above, defines a case in which summaries are lossless. That is, given a tree summary, it is possible to reconstruct the original values at the lattice leaves *exactly*, by looking up the closest weighted ancestor for each leaf.

3.2.2 Lossy Case

While in some cases it is preferable to have a lossless summary, it usually comes at a high cost (i.e., the size of the summary). To mitigate the cost we

introduce lossy summaries that trade reconstruction accuracy for conciseness. To do that we use the notion of an *α-interval*. We define a summary tree T to be an α-approximate lossy summary of a lattice L, $\alpha \in [0, 1]$, provided the following conditions are satisfied:

- For every leaf t of T, the weight of its closest weighted ancestor in T is within the interval $[(1 - \alpha) \cdot val(t), (1 + \alpha) \cdot val(t)]$, where $val(t)$ is the original value associated with the leaf (cell).

Notice that the choice of α is orthogonal to our problem. In principle, each cell could have a different α value, based on its regular variation, seasonality, etc. Lossless summaries correspond to the special case where $\alpha = 0$.

An α-approximate lossy summary naturally provides a bound on the amount of information loss per node. We will establish the bound after introducing the necessary reconstruction error metric in Section 6.2. The cost of a lossy summary is defined in the same way as lossless summaries, i.e., the number of tree nodes which have a weight associated with them.

The problem we study is, given a multidimensional dataset with corresponding lattice structure L and a number $\alpha \in [0, 1]$, find an α-approximate summary of L with the least possible cost.

Chapter 4

Algorithm

In this section we present our TS (Tree Summary) algorithm. The algorithm runs in two steps. The first step finds promising weights for each of the internal nodes of the lattice. Then, that information is used in the second step to find the most concise summary.

4.1 Node Annotation

In order to keep track of what weights are best, we annotate all nodes in a lattice. An *annotation* of a node $\vec{p}$ is a tuple (w, Dim_j, c), where w is a weight, Dim_j is a dimension node of the node $\vec{p}$, and c is the cost corresponding to w and Dim_j. The meaning of every annotation at node $\vec{p}$ is as follows. A subtree rooted at $\vec{p}$ could be assigned a weight of w and expanded along dimension j with the total minimum cost of the subtree of c. Each node can have multiple annotations; they will be referred to as annotation sets.

Precomputing annotations is the first step in finding a tree summary. Annotations for leaf nodes are trivial. The proposed weight of a leaf node is simply its cell value, the cost is 1, and since leaves have no children, the dimension node is none. Annotations for the internal nodes are computed iteratively bottom up, by finding the most popular weight(s) among each node's children across all dimensions. The process is described in more detail in Algorithm 1 below.

Here, $weights(a_{\vec{p}})$ is the set of weights in annotations of $\vec{p}$ and $cost(a_{\vec{p}})$ is the cost of every $w \in weights(a_{\vec{p}})$. Note, that we only propagate minimum cost weights, so all weights in the set of annotations have the same cost. On line 12, we find the weight w that can be found in annotations of the largest number of children of $\vec{p}$ in the given dimension. Line 13 calculates the cost associated with w; since root is always present in the summary we charge a cost of 1 for the root of the subtree, and add the cost of the children nodes, given that their parent's weight is w. Finally, at line 16, out of all annotations $a_{\vec{p}}$ the $filter(\cdot)$ function only leaves annotations with smallest cost. We run Algorithm 1 by calling `annotate(`$\vec{r}$`)`, where $\vec{r}$ is the root of

Algorithm 1 The annotation algorithm `annotate(`$\vec{p}$`)`

Input: Lattice node $\vec{p}$

if $\vec{p}$ is a cell **then**
  Let $a_{\vec{p}}$ be the set of annotations of $\vec{p}$
  $a_{\vec{p}} \leftarrow \{(val(\vec{p}), \varnothing, 1)\}$
else
  Let D be the set of dimension nodes of p
  for each Dim_j in D **do**
    $W = \varnothing$
    for each child $\vec{q}$ of Dim_j **do**
      `annotate(`$\vec{q}$`)`
      $W = W \cup \{weights(a_{\vec{q}})\}$
    end for
    $w = \text{argmin}_{w \in W} |\{\vec{q} \in Dim_j : w \notin weights(a_{\vec{q}})\}|$
    $c = 1 + \sum_{\vec{q} \in Dim_j} (cost(a_{\vec{q}}) - [w \in weights(a_{\vec{q}})])$
    $a_{\vec{p}} \leftarrow a_{\vec{p}} \cup (w, Dim_j, c)$
  end for
  $a_{\vec{p}} \leftarrow filter(a_{\vec{p}})$
end if

the lattice; the annotations of all nodes are then found recursively bottom up.

4.2 Summary Construction

Algorithm 2 constructs a minimum weight summary by using the precalculated annotations from the previous step. We run the algorithm by calling `summarize(`$\vec{r}, t$`)`, where $\vec{r}$ is the root of the previously annotated lattice and t is the root of the future summary tree, which at the start is just a single node.

Algorithm 2 The algorithm for finding an optimal summary tree `summarize(`$\vec{p}$`,` t`)`

Input: Lattice node $\vec{p}$, tree node t

1: **if** $\vec{p}$ is the root of a lattice **then**
2: Pick any $\hat{a} \in a_{\vec{p}}$
3: Let w_t be the assigned weight of t.
4: $w_t = \hat{a}.weight$
5: **else**
6: Let w' be the weight of the closest weighted ancestor of t.
7: **if** $\exists \hat{a} \in a_{\vec{p}}$, s.t. $w' = \hat{a}.weight$ **then**
8: $w_t = \varnothing$
9: **else**
10: Pick any $\hat{a} \in a_{\vec{p}}$
11: $w_t = \hat{a}.weight$
12: **end if**
13: **end if**
14: **for** each child $\vec{q}$ of $Dim_j, j = \hat{a}.dim$ **do**
15: Add a new node t_{child} as a child of t
16: `summarize(`$\vec{q}, t_{child}$`)`
17: **end for**

Algorithm 2 starts with the lattice root and works its way top down. It chooses any annotation at the lattice root node and assigns that annotation's weight as the weight of the tree summary root (lines 2-4). For all other nodes, we pick an annotation whose weight matches the weight of the closest ancestor (line 7), and if no such annotation exists any other annotation is chosen (line 10). Weight assignment for the current tree node happens on lines 8 and 11. Finally, for every child $\vec{q}$ of the lattice node $\vec{p}$ in the dimension

$\hat{a}.dim$, we create a tree child t_{child}, and recursively do weight assignment on its descendants (line 16).

4.3 Example

Now, let's consider the lattice from our earlier equipment store example. We show a detailed algorithm walk-through on the sublattice corresponding to "Camping Equipment, All Stores". First, we run the bottom-up annotating algorithm. At the leaves, we will have the following annotations:

"Sleeping Bags, Manhattan"	$\{(1.1, \varnothing, 1)\}$
"Tents, Manhattan"	$\{(1.1, \varnothing, 1)\}$
"Sleeping Bags, Queens"	$\{(0.8, \varnothing, 1)\}$
"Tents, Queens"	$\{(0.8, \varnothing, 1)\}$
"Sleeping Bags, Newark"	$\{(0.9, \varnothing, 1)\}$
"Tents, Newark"	$\{(1.0, \varnothing, 1)\}$

For every internal node, Algorithm 1 finds the most popular weight(s) among the node's children annotations for every drill-down, and only leaves the annotations that result into least cost for that node's subtree. At the next level of the lattice the annotations are as follows:

"Camping Equipment, Manhattan"	$\{(1.1, Dim_P, 1)\}$
"Camping Equipment, Queens"	$\{(0.8, Dim_P, 1)\}$
"Sleeping Bags, NYC"	$\{(1.1, Dim_L, 2), (0.8, Dim_L, 2)\}$
"Tents, NYC"	$\{(1.1, Dim_L, 2), (0.8, Dim_L, 2)\}$
"Camping Equipment, Newark"	$\{(1.0, Dim_P, 2), (0.9, Dim_P, 2)\}$

Next, let's calculate annotations for the node *"Camping Equipment, NYC"* which is expandable in both dimensions. When we expand that node along Dim_P, the size of the summary is 3; when we use Dim_L instead, the size is 2 with the weights 1.1 and 0.8 being equally good. So, we have $\{(1.1, Dim_L, 2), (0.8, Dim_L, 2)\}$ for *"Camping Equipment, NYC"*. At the node *"Camping Equipment, All Stores"*, there is a tie between four annotations $\{(0.8, Dim_L, 4), (0.9, Dim_L, 4), (1.0, Dim_L, 4), (1.1, Dim_L, 4)\}$, all of them resulting in the same cost. It is important, however, to save all four weights because without additional knowledge about the rest of the lattice, it is impossible to evaluate which weight will ultimately lead to the least cost of the summary.

By repeating the same annotation procedure on the rest of the lattice, we obtain the annotation for the second half of the lattice (*"Outdoor Protection, All Stores"*) which is $\{(1.0, Dim_P, 4)\}$. Since the weight of 1.0 is included in both nodes ("Camping Equipment, All Stores" and "Outdoor Protection, All Stores"), at the root we will have $\{(1.0, Dim_P, 7)\}$, with the weight of 1.0 breaking the tie between all possible weights for the node *"Camping Equipment, NYC"*. If we calculate annotations for the other drill-down from root, namely *"All Products, NYC"* and *"All Products, Newark"* internal nodes, we will discover that indeed Dim_P is the best dimension that the root can be expanded on.

Now that we have found annotations for each node, we can start constructing the actual summary tree. We use the root's annotation $\{(1.0, Dim_P, 7)\}$, assign the weight of 1.0 to the root and split on the product dimension. Going back to the node *"Camping Equipment, All Stores"*, we do not assign any weight to it because the weight of the root (i.e., the closest weighted ancestor) is already equal to one of the annotation weights of this node. Whenever this condition does not hold, we override the weight of the closest ancestor by assigning any weight from available node annotations. We repeat the process for all remaining nodes. The output of Algorithm 2 is shown in Figure 1.2. In Figure 3.1, weighted nodes of the summary tree are highlighted in yellow.

4.4 Lossy Case

The algorithms remain generally the same for the lossy case, except for a few minor modifications. Both algorithms 1 and 2 will have to be changed accordingly. For example, w will now have to refer to the confidence interval, instead of a single weight, and similarly, whenever we compare two weights or check the membership of a weight in a set, we now seek the intersection of the two weight intervals. In Algorithm 1, line 3 changes to $a_{\vec{p}} \leftarrow \{([(1-\alpha) \cdot val(\vec{p}), (1+\alpha) \cdot val(\vec{p})], \varnothing, 1)\}$. In Algorithm 2, whenever given a choice, we pick a middle point of an interval as the weight. So, $\hat{a}.weight$ is replaced by $(w_{start} + w_{end})/2$, where $\hat{a}.weight = [w_{start}, w_{end}]$.

Chapter 5

Theoretical Analysis

5.1 Run-time Analysis

Denote the number of nodes in the lattice by n and the number of dimensions by d. Algorithm 1 finds annotations for n nodes (we only calculate annotation sets for each node once) in each of the d dimensions. For each node, the algorithm iterates through the node's set of annotations to filter out annotations with a suboptimal cost. In the worst case scenario, the number of annotations is equal to the number of leaf descendants of the node which is $< n$. Therefore, `annotate(·)` runs in $O(n^2 d)$ time.

Now, let's analyze the run time of Algorithm 2. We only iterate through k nodes, since only $k < n$ of the lattice nodes get considered during the tree summary construction phase, where k is essentially the number of nodes in the future summary tree. Then, for each node, the algorithm iterates through the node's annotations to find the best annotation given a parent's weight. As noted previously, the worst case number of annotations per node is the number of that node's leaf descendants and that is $< n$, so the algorithm `summarize(·)` runs in $O(n^2)$ time.

5.2 Optimality Analysis

To show that the algorithm provides an optimal node weighting, we start with presenting the blocking property of a tree data structure in the context of weight assignment. In particular, we show that to find an optimal weight (or a set of weights) of any internal node in a tree, it is sufficient to consider weights (as provided by TS algorithm) of the tree's immediate children only.

For clarity, what we refer to as a weighting scheme, is an embedded summary tree in the lattice (i.e., an embedded tree where a selected set of nodes is assigned weights).

Lemma 5.2.1. ***(Blocking Property)*** *Let T' be any subtree embedded in the lattice. Let T be a supertree in the lattice which contains T' as a subtree. Let $\mathcal{O}$ be the set of optimal node weightings of T and $\mathcal{S}$ be the set of weightings*

of T' found by TS algorithm. Then, there exists a weighting scheme $S \in \mathcal{S}$, such that the weighting O^{**} obtained by changing $O^* \in \mathcal{O}$ so that it agrees with S on nodes in T', has no more cost than O^*.

Proof. We show the above property by contradiction. Assume that there is no such $S \in \mathcal{S}$, and that O^{**} has a larger cost than O^*. Since T' is the only subtree that is different in O^{**} and O^*, $cost(T'_S) > cost(T'_{O^*}), \forall S \in \mathcal{S}$. In other words, the cost of any weighting scheme S on T' is strictly greater than the cost of the optimal weighting on T'. Since TS propagates the most popular weights, there exists some node q for which the optimal weight was not the weight of the majority of q's children.

Let p be q's parent, by n let's denote the number of q's children, and by k we use the number of children with the most popular weight(s). Assume that $\{x\}$ and $\{y\}$ are the most popular weights. Let's also denote a number of children of some arbitrary weight $\{z\}$ by m. Refer to the Figure 5.1.

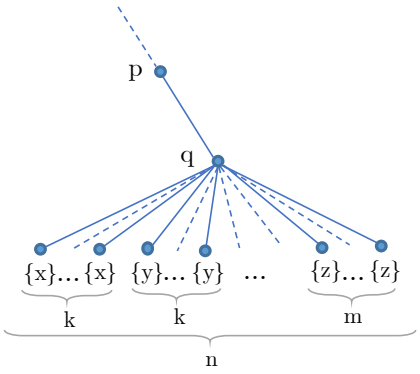

Figure 5.1: Structure of the node q

The TS algorithm will choose any of the most popular weights x or y. When $w_p = x$ (or y), the weight of node q is $w_q = 0$ and the total cost of q and its children combined is $cost(q_S) = n - k$. In the opposite case, if $w_p \neq x$ (and $w_p \neq y$), w_q will be set to $x - w_p$ (or $y - w_p$), and the total cost of q's subtree is $cost(q_S) = n - k + 1$.

We assumed that in O^*, q was assigned a weight different from all $S \in \mathcal{S}$. Let's denote that weight by z. The total cost of q's subtree is $cost(q_{O^*}) = n - m$ if $w_p = z$, and $cost(q_{O^*}) = n - m + 1$ if $w_p \neq z$.

For any of the weights chosen by $S \in \mathcal{S}$, $k > m$. Therefore, $cost(q_S) \leq cost(q_{O^*})$. That means that TS performs at least as well as optimal; however, by our assumption, $cost(S_{T'}) > cost(O^*_{T'}), \forall S \in \mathcal{S}$, and that leads to a

contradiction. Hence, $\exists S \in \mathcal{S}$, such that O^* agrees with S on subtree q. □

Theorem 5.2.2. ***(Optimality)*** *Let O^* be some optimal weighting of a lattice and $\mathcal{S}$ denote the set of TS weightings. Then for every node q in O^*, $\exists S \in \mathcal{S}$, such that if we replace the subtree of q_{O^*} with q_S the cost of O^* does not increase.*

Proof. Let's assume that the claim is false and $\exists q, \forall S \in \mathcal{S}$, so that replacing q_{O^*} with q_S results into a more expensive summary. It means there is some node q for which O^* chose a different weight and/or dimension node, than any $S \in \mathcal{S}$.

Let's compare nodes in O^* to those in S in the order of their shallowness. We start at the root and go down the tree until we find the shallowest node q that is different in both summaries. Assume that for summary node q the corresponding lattice node is $\vec{q}$ and its annotations are as follows: $\vec{q} : \{(w_S, Dim_S, cost(\vec{q}_S)), (w_{O^*}, Dim_{O^*}, cost(\vec{q}_{O^*}), ...\}$, where subscript S refers to the TS choice, and subscript O^* corresponds to the optimal choice.

TS stores all annotations of least cost (line 16 in Algorithm 1) and among those it favors the annotations with the same weight as the weight of the closest weighted ancestor (line 7 in Algorithm 2). This implies that there are two possibilities for the optimal solution $a_{O^*} = (w_{O^*}, Dim_{O^*}, cost(\vec{q}_{O^*})$ to not be included among TS annotations:

- a_{O^*} is not in $\vec{q}$'s annotations, meaning that $cost(\vec{q}_S) < cost(\vec{q}_{O^*})$. In that case, any weight w_S is guaranteed to be at least as good as w_{O^*}, regardless of the weight of the closest weighted ancestor (which could be w_S, w_{O^*}, or neither).

- a_{O^*} is among $\vec{q}$'s annotations but did not get chosen by TS. Since TS picks w_S that equals the closest weighted ancestor's weight whenever possible, any weight w_S is guaranteed to result in at most the same cost as w_{O^*}.

So, replacing q_{O^*} by q_S doesn't result in increasing cost. Thus, for any node q, $\exists S \in \mathcal{S}$, such that O^* agrees with S on weighting scheme of q. □

Chapter 6

Empirical Evaluation

In this section we assess the performance of the proposed algorithm using real and synthetic data. We evaluate the accuracy of produced summaries by calculating the reconstruction error, and confirm the worst-case error bound (to be shown theoretically in Section 6.2). We compare the performance of our algorithm with four baselines under varying settings, such as size, number of dimensions and the amount of correlation in the data.

6.1 Baselines

In addition to TS, we run experiments on 4 baselines. The first two are the approach by Karloff et al. (which we will denote by K) and the Cascading Analysts (CA) algorithm. The third baseline is STCK-T which consists of trees created by stacking dimensions in an exhaustive manner, and then the average metrics are reported. The last baseline is RND-T that chooses 100 random trees embedded in the lattice; similarly we report the averages. After constructing the trees in the last two baselines, we run the algorithm by Agarwal et al. [1], which finds optimal summaries for single-dimensional data.

Since CA needs both positive and negative values to operate, instead of assigning ratios between two data snapshots to each leaf node, we assign a logarithm of a ratio instead. During the reconstruction phase, we translate the values back by taking an exponent of the reported values.

6.2 Measuring Reconstruction Error

To measure the accuracy of the constructed summaries we reconstruct the original lattice cell values in the following way. For TS, the value of each cell is reconstructed by taking the weight of the closest weighted ancestor in the tree summary. For RND-T and STCK-T, a cell value is calculated by taking a sum of weights along the root-to-leaf path of the corresponding leaf node in the summary tree. For K, each cell is assigned the value equal

to the sum of all nodes (rectangles) that the cell is a descendant of. For CA, since each node included in the summary contains an accumulated sum of that node's leaf descendants, we divide each summary node's value by the number of leaves it is an ancestor of, and assign that value to each of those cells.

Let l denote a leaf node in the lattice, and let l_{orig} and l_{rec} correspond to l's original and reconstructed values, respectively. We measure the accuracy of reconstruction by symmetric mean absolute percentage error (SMAPE), as described in [5]:

$$error = \frac{1}{|L|} \sum_{l \in L} \frac{|l_{rec} - l_{orig}|}{|l_{rec}| + |l_{orig}|},$$

where L is a set of all lattice cells. We have chosen this error metric, because of its desirable properties of symmetry, bounded range and interpretability due to its similarity with percentage metrics.

Now, having definitions of a confidence interval and the error metric, it is straightforward to see that for TS, the amount of error per node is bounded and can be expressed through α. In the worst case, when a node's closest weighted ancestor's weight is equal to one of the end points of the confidence interval, the error $error_p$ of node p is expressed as:

$$error_p = \frac{|val(p) \cdot (1 - \alpha) - val(p)|}{|val(p) \cdot (1 - \alpha)| + |val(p)|} = \frac{\alpha}{2 - \alpha}.$$

In the similar fashion, we show the same bound for the other end of the confidence interval $(1 + \alpha) \cdot val(p)$.

6.3 Setup

Implementation of our algorithm and the alternative approaches by Karloff et al. [8], Ruhl et al. [13] and Agarwal et al. [1], are coded in Java 8. Experiments have been run on a 3.50 GHz Windows machine with 16 GB of RAM.

6.4 Datasets

6.4.1 US Census Educational Attainment Data

The first dataset is by the US Census Bureau on levels of educational attainment for different demographic groups 18 years old and over, across the

United States [19]. It has four dimensions: geography, sex, age and level of education. Geography is a hierarchy with 4 levels: entire country, 4 regions, 9 divisions and 50 states. All three of sex, age and education dimensions, are two-level hierarchies, consisting of a root and leaf nodes. Sex has 2 values, age has 5 distinct age groups and education contains 7 education levels. We are using the 2016 and 2017 data, and summarize the change between the two snapshots as ratios of values between the two years.

6.4.2 Synthetic Data

The rest of the test data is generated synthetically. First, we generate tree hierarchies for each dimension by starting at the root and making a random decision on whether or not a node has children, then we draw a number of children from a normal distribution ($\mu = 5, \sigma = 2$). We do so iteratively until no more nodes are generated. Given the generated hierarchies we build a lattice as a cross-product of trees.

For both the lossless and lossy case, we first find a random embedded tree in a lattice and assign a value of 1 to the root. Then we go top down, and each child q of a node p, is assigned a value drawn from a normal distribution with the mean μ being set to the parent's value assigned in the previous step and a preset standard deviation $\sigma = 0.1$. The values at the leaf nodes of the embedded tree are what is later assigned as lattice leaf values.

6.5 Results

The algorithm by Karloff et al. is constrained to only 2 dimensions. Since randomization is used by the algorithm, all results are averaged across 100 independent runs. Similarly, RND-T results are averaged across 100 runs, and STCK-T results are averaged across all possible permutations of dimension stacking. Unless otherwise stated, CA budget size k is set to be equal to the size of the TS summary and is thus excluded from the size comparison.[2]

6.5.1 Lossless Case

We start with testing the performance and efficiency of all three approaches in a lossless case. We run TS and Karloff's et al. approach with $\alpha = 0$. To test CA, we run it two times: first we give it a budget of k that is equal to the size of the tree summary, and the second time we set k to be equal to the number of leaf nodes.

[2] In almost all cases, the final size of CA summary is equivalent to that of TS.

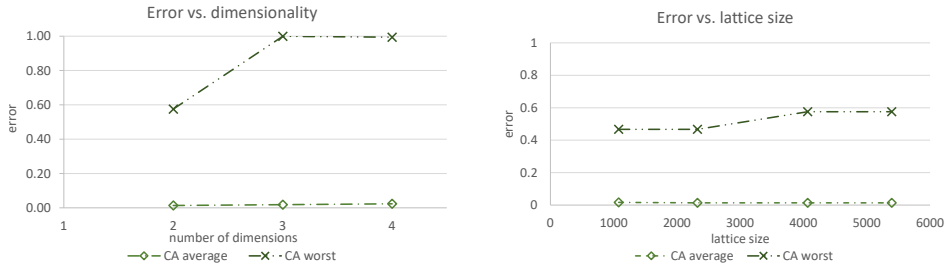

Figure 6.1: US Census dataset. Reconstruction error in lossless case

US Census data. First, we perform experiments on the US Census Educational Attainment table. The number of dimensions was varied by stacking demographics dimensions (sex, age and education). The size of the lattice was varied with the size of the geographic dimension that was changed to include either one, two, three or all four of the US regions. When varying the size, the number of dimensions was locked at 2. Since it is impractical to summarize continuous range data in a lossless setting, we round all values to one decimal place.

We measured the reconstruction errors across all approaches. The errors are shown in Figure 6.1. TS, K, and both tree baselines have a 0 reconstruction error and are not included in the plot. We confirmed that CA could only find lossy summaries in both cases, when given a budget that is equal to the TS summary size and the number of all lattice cells. We therefore omit CA from further comparisons on a lossless case.

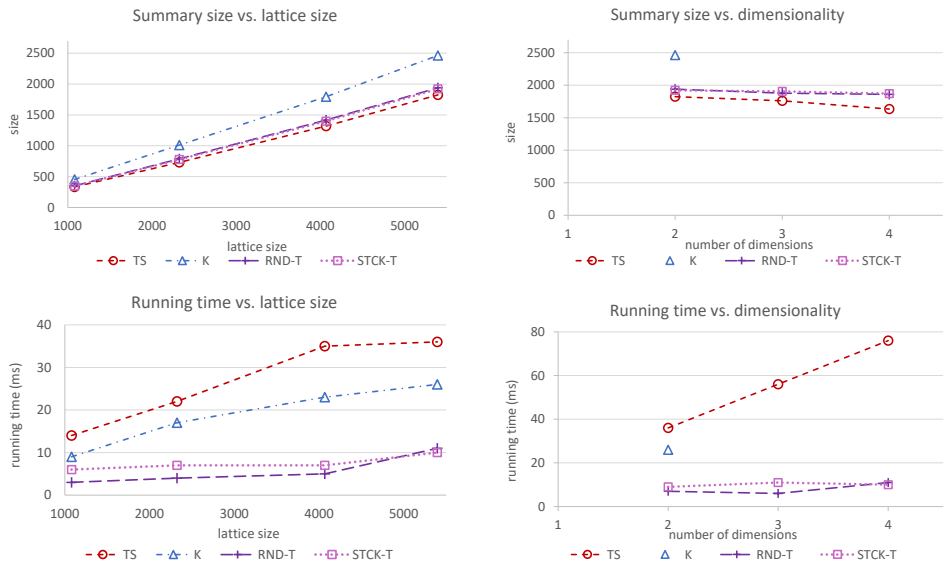

Figure 6.2: US Census dataset. Summary size and running time in a lossless case

In Figure 6.2, we compare TS to the remaining three approaches. An interesting observation is that TS outperforms Karloff's et al. in terms of the summary size. In their paper they have shown that their algorithm is a randomized 2-approximation from the optimal. So empirically, TS appears to achieve a more concise hierarchical summary although its space is more restrictive (allowing no overlaps except containment) than that of Karloff et al.'s. When varying lattice size and number of dimensions, TS finds more concise summaries than all of the other approaches, however it does come with a higher execution cost.

Synthetic data. Next, we test TS and the three baselines on synthetic data. The data was generated as described in Section 6.4.2. As previously, the values have been rounded to one decimal place.

Besides varying the size and number of dimensions, we also test how the algorithms perform in the presence of correlation. The idea is that in deeper and more granular levels of the lattice, the entities are more similar to each other. For example, given that San Francisco and San Jose are located in the same geographical region, the sales of sunscreen at the two stores may be more similar to each other than to the sales of sunscreen at a New York store. To simulate this correlation, we reduce the value of the standard

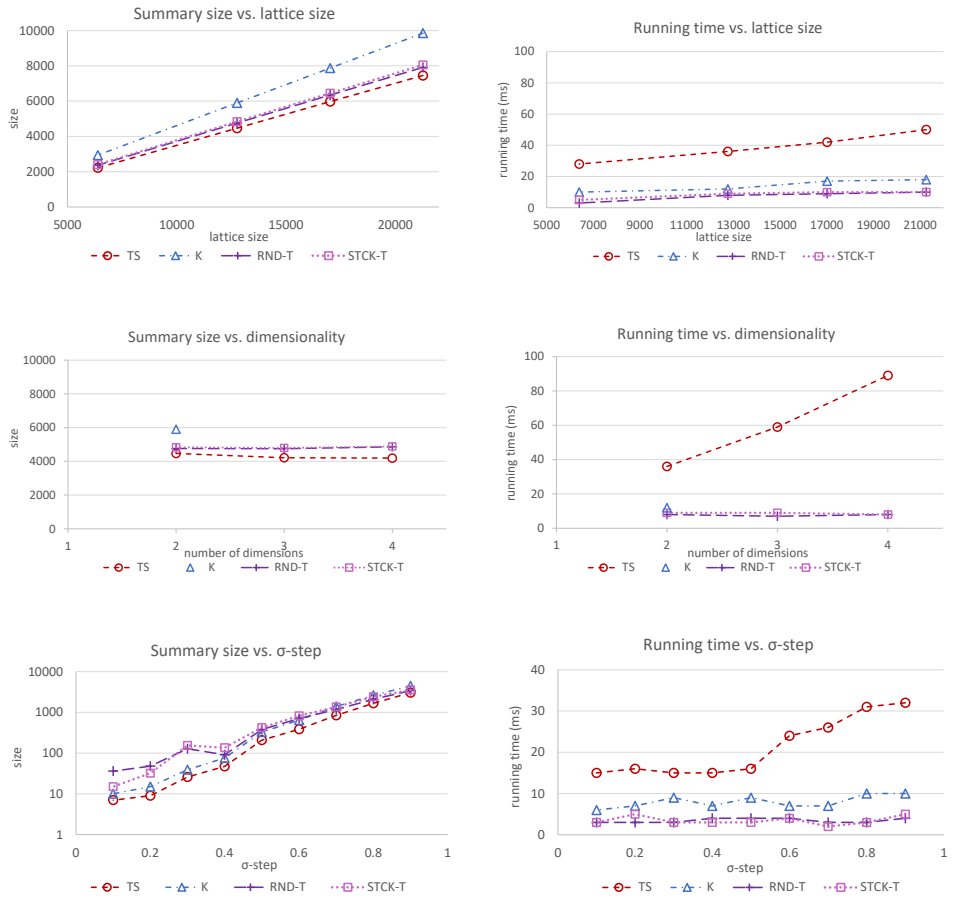

Figure 6.3: Synthetic dataset. Summary size and running time in a lossless case.

deviation as we go deeper in the embedded tree. For a parent node p with SD σ_p and its child node q, $\sigma_q = \sigma_p \times \sigma\text{-}step$, where $0 < \sigma\text{-}step \leq 1$.

The results are shown in Figure 6.3. Although TS computes in a reasonable time (in the order of milliseconds), it takes longer than other baselines. It does, however, find more concise summaries across all runs.

6.5.2 Lossy Case

Here, we test how the algorithms perform in a lossy setting, trading the accuracy of reconstruction for conciseness. First we see how the summary size changes when varying the α parameter.

As mentioned earlier, the work by Karloff et al. [8] is only designed for lossless summaries. However, it relies on the algorithm introduced by Agarwal et al. [1] that solves the same problem of building hierarchical summaries but in one-dimensional space. Agarwal et al. do have a natural extension that allows for lossy but more concise summaries. An obvious straightforward way to extend Karloff et al.'s work to produce lossy summaries, is to use their base framework together with the extended algorithm from Agarwal et al. However, no guarantees were presented in the original paper. We compare TS with the modified Karloff's et al. approach on a range of alpha values in Figure 6.4. We can confirm with our empirical results that no previous guarantees for the lossless case, hold anymore for the lossy case in the case of that simple extension, and that the summary size for K across all runs was significantly larger and thus incomparable to other approaches. For this reason, we eliminate the K baseline from lossy experiments.

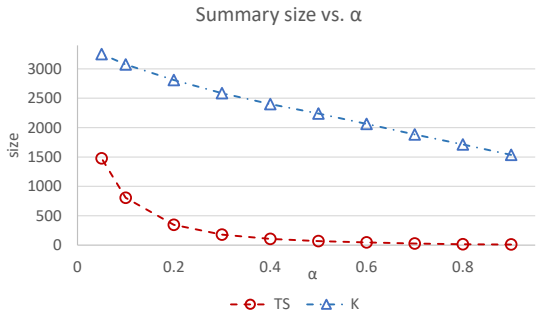

Figure 6.4: US Census dataset. Comparing TS and K in lossy summaries.

US Census data. First we vary the parameter α. The results are shown in Figure 6.5. The summary size of TS is slightly smaller than the tree baselines. CA is best in terms of average error, but it needs significantly more time than any other method; it also does not perform well in terms of the worst case error. TS and tree baselines have comparable average and worst case errors. Note, that all three optimize for the worst case error and have the same error bound guarantees; worst case error coincided across all runs (up to 3-4 decimal places) and is thus reported with a single line in the graphs.

We fix $\alpha = 0.1$ and proceed comparing all four approaches in a lossy setting in Figure 6.6, for varying lattice size and number of dimensions. TS finds smaller summaries in all cases, and it improves its summary size when increasing the number of dimensions. Although the size of CA is exactly the

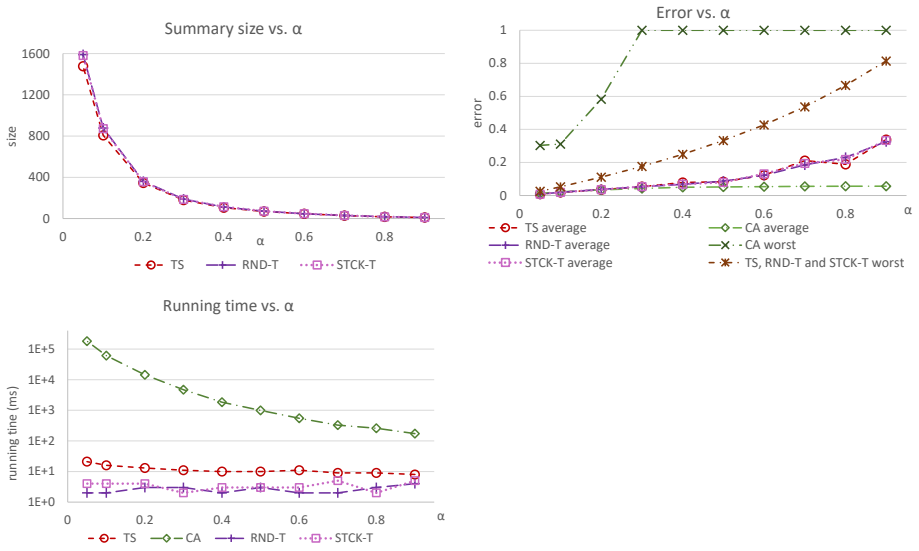

Figure 6.5: US Census dataset. Summary size, reconstruction error and running time in a lossy case with different values of α.

same as that of TS, CA has larger errors and needs a considerably longer execution time.

Synthetic data experiments are performed on the same synthetic datasets as in the lossless case, but we use the generated data values directly, without rounding. First we vary parameter α and show the results in Figure 6.7, and then we fix the value of $\alpha = 0.1$ and test the performance against lattice size and number of dimensions in Figure 6.8. Since for a correlated case we can expect smaller summaries, we changed α to 0.05 for more accurate reconstruction. The results are shown in Figure 6.11.

Across all runs, TS is showing the best performance in terms of the summary size. Since the summary size of TS is equal to the budget of CA, we compare the two on the basis of reconstruction errors and running time. In a few cases, particularly when α is set to bigger values, CA obtains a better performance than TS. This is explained by the fact that with a broader confidence interval, any value within the interval is treated the same and can be assigned as weights, although the values closer to the interval ends are clearly of a lower quality. As we reduce α, TS starts to outperform CA in terms of both average and worst case errors. To better understand the error distribution of the two, we show the error histogram for the case when $\alpha = 0.1$ in Figure 6.9. TS is also significantly faster than CA in all

runs.

TS and tree baselines RND-T and STCK-T have the same parameter α and an identical worst-case error bound. The three are mainly compared on the basis of summary size and running time. TS has a bigger search space and thus runs slower but succeeds at finding more concise summaries in all cases. To have a clearer picture of the distributions of summary sizes for TS, RND-T and STCK-T, we show a size histogram for a 4D case in Figure 6.10. Although all three methods share the same worst-case bound, in a few cases the average error of the tree baselines is slightly lower than that of TS. The intuition to this phenomenon is that given the same worst-case bound, a bigger summary (as found by RND-T and STCK-T) contains more information and results in a smaller average error during reconstruction.

6.5.3 Scalability

Typically, even a snapshot of a real world multidimensional warehouse contains data of higher dimensionality and size. In this section, we are testing the TS algorithm on a larger lattice (Figure 6.12).

We generated a 6D dataset in a similar fashion as earlier ($\sigma = 0.1$). The generated lattice has nearly 5M nodes, among which 1M are leaf nodes. TS exhibits a similar behavior to that in smaller examples. The summary size grows significantly as we decrease α. The running time is not affected as much. As α decreases, fewer nodes can be grouped under the same annotation, and as a result, more annotations are created but it results in only a slight increase in running time.

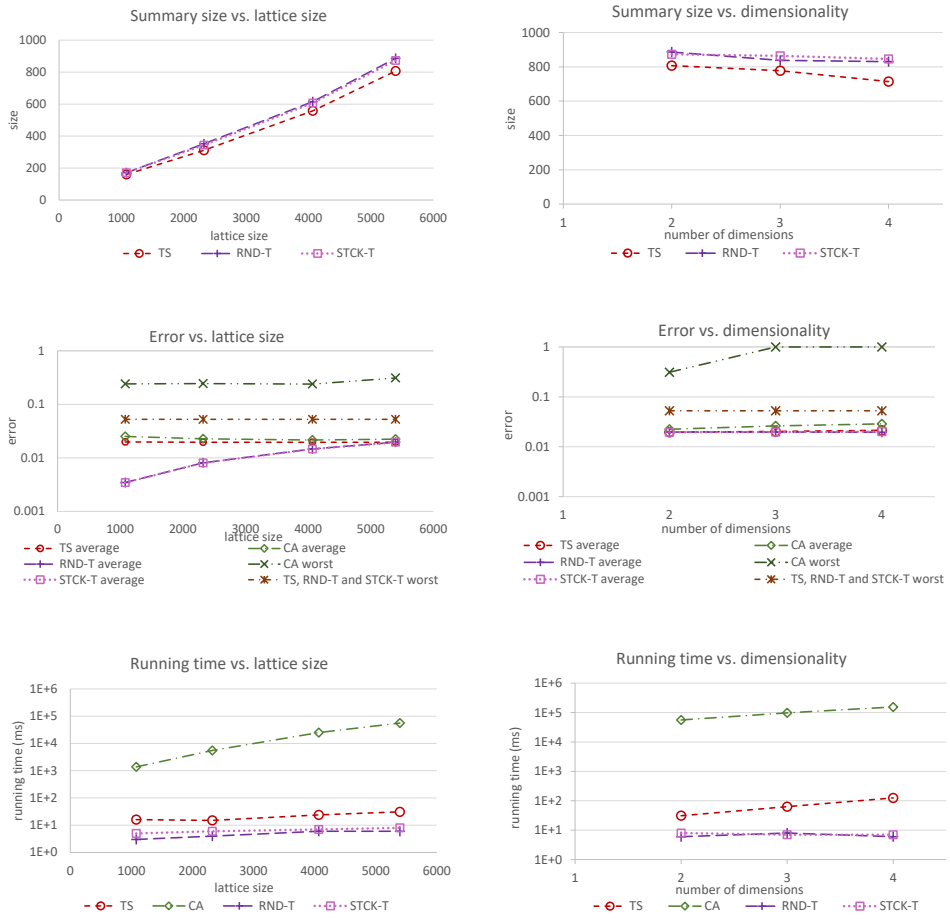

Figure 6.6: US Census dataset. Summary size, reconstruction error and running time in a lossy case, $\alpha = 0.1$.

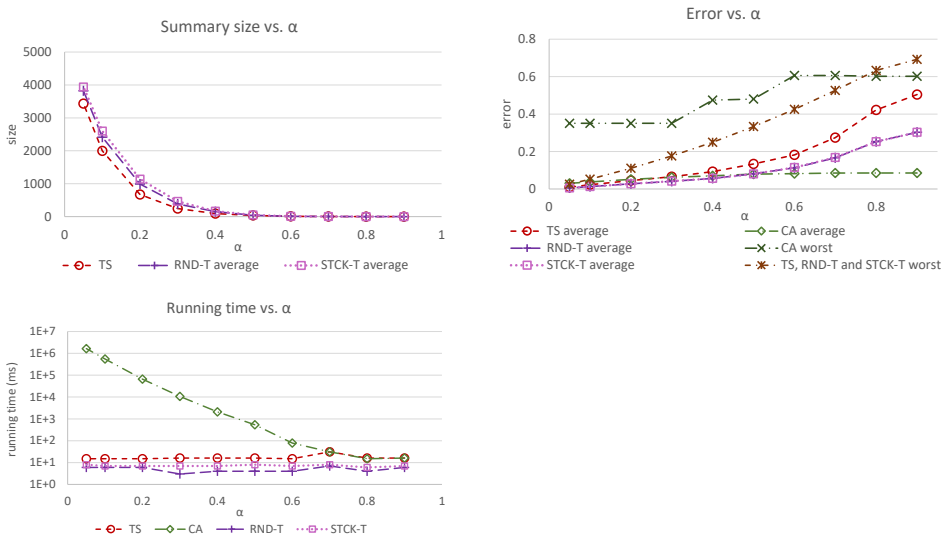

Figure 6.7: Synthetic dataset. Summary size, reconstruction error and running time in a lossy case with different values of α.

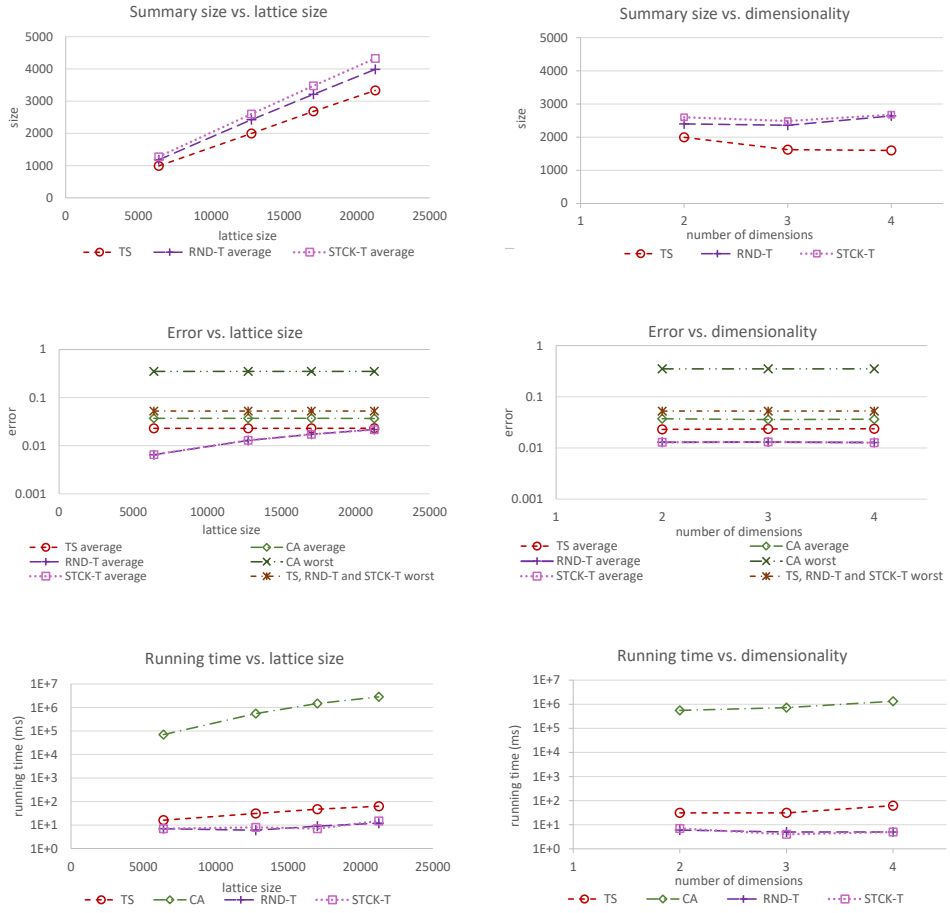

Figure 6.8: Synthetic dataset. Summary size, reconstruction error and running time in a lossy case, $\alpha = 0.1$.

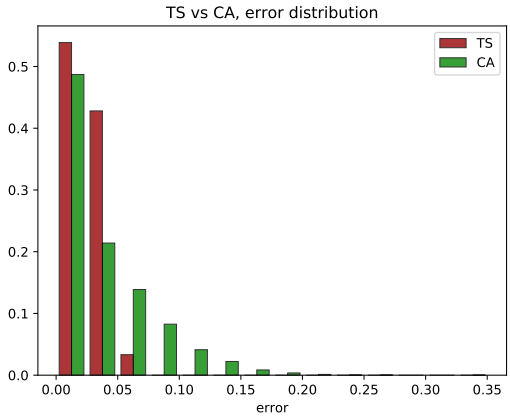

Figure 6.9: Normalized error histogram of TS and CA for synthetic lossy experiments with $\alpha = 0.1$.

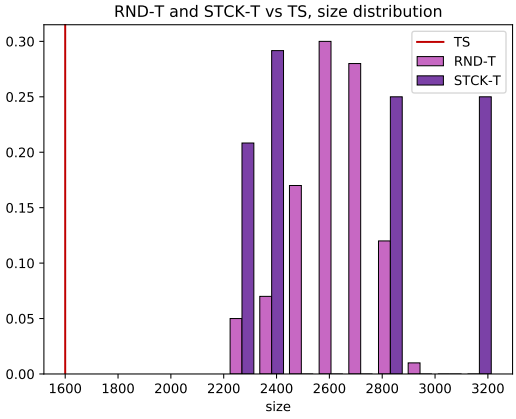

Figure 6.10: Normalized summary size histogram for RND-T and STCK-T, for synthetic lossy experiments with $\alpha = 0.1$ in 4D.

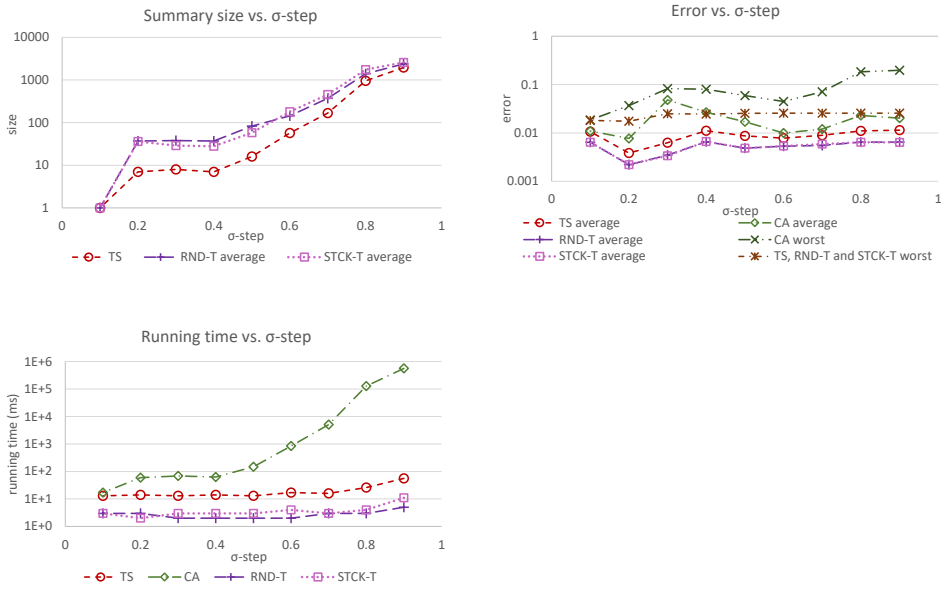

Figure 6.11: Synthetic dataset. Summary size, reconstruction error and running time in a correlated lossy case, $\alpha = 0.05$.

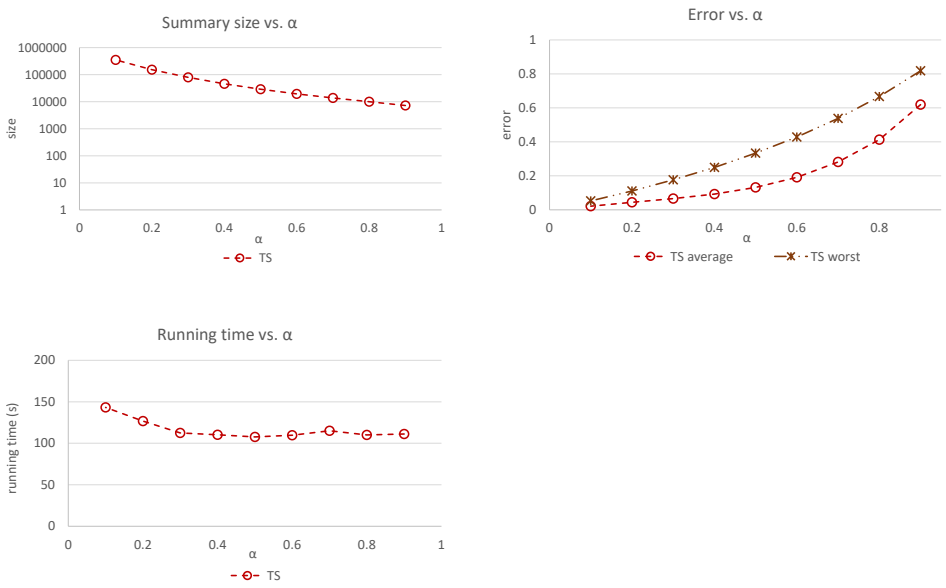

Figure 6.12: Large synthetic dataset, 5M nodes, 1M leaves. Summary size, reconstruction error and running time in a lossy case with different values of α.

Chapter 7

Discussion

7.1 TS Summary Options

In our study, we have chosen to summarize changes as ratios between values in two snapshots because of the natural normalization which ratios provide. For example, if the sales of item 1 at a particular store increased from 100 to 150, and the sales of item 2 increased from 1000 to 1500, the change of both could be summarized as 1.5. While using differences instead of ratios may not be favorable in this scenario, it may be more favorable in other cases, e.g., when a user is interested in the absolute change. Similarly, instead of ratios one could choose to use percentage change or some other change metric. This provides some flexibility for an analyst to choose what feels more natural.

7.2 Missing Values

When comparing changes between two snapshots, oftentimes there can be cases of missing values in one or both of the snapshots. That might happen, for example, if a store stops selling a particular item, or conversely, introduces a new item to the market. Since it is a very common use case, we need to specify a mechanism for dealing with missing values.

When values are missing from both snapshots the corresponding summary node is simply absent in the summary tree and therefore does not affect the tree. However, when only a single value is missing from a pair of snapshots it is not clear how change should be calculated. The summary, however, should still contain information about what values are missing from the data to present a complete summary. This problem can be addressed by assigning special values to the nodes with missing information. The special values cannot be grouped under any confidence intervals but only by the same special value. The propagation will then occur in the same way as described in Section 4. This is a simple way to deal with sparse data. Since, empirically, sparsity hasn't shown any interesting insights and none

of the other approaches have described a way to deal with sparsity, we omit experiments performed on data with missing values.

7.3 Multiple Measures Propagation

There are situations in which one may want to track more than one measure. For example, an analyst may be interested in a summary that contains both temperature and pressure or wind speed, to see if there are any interesting patterns. Similarly, if an analyst looks at the outcomes of some Internet advertisement, perhaps they would be interested not only in the number of ad clicks per demographic group, but the number of times a product was sold after users clicked on the ad.

In that case, we could propagate weights for all measures simultaneously. The definition of the cost function may be changed as below:

$$cost(T) = \sum_{t \in T} \sum_{m=1}^{M} [w_{t_m} \neq \varnothing],$$

where M is the number of measures and w_{t_m} is a weight of a tree node t for measure m.

Instead of charging a unit cost for every measure in a node whose weight is not 0, another possibility is to charge a cost for each node that has at least one non-zero weight measure, as defined below. As opposed to the previous cost function, this cost function encourages measures to find the best weights jointly, potentially allowing us to see interesting patterns between measures that could have not arisen previously.

$$cost(T) = \sum_{t \in T} (\bigvee_{m=1}^{M} [w_{t_m} \neq \varnothing]).$$

Both approaches require more exploration to understand the pros and cons of using joint versus individual summaries and to see if joint summaries of multiple measures enable users to see other patterns, such as correlation between measures.

Chapter 8

Conclusions

In this work we study the problem of summarizing multidimensional data. We proposed an efficient algorithm (TS) that finds optimal tree-embedded summaries in polynomial time. Our algorithm makes an effective use of the hierarchical structure of the data by distributing weights along the hierarchies and defining values at the leaf nodes as weights of their closest weighted ancestors.

We evaluated our approach on real and synthetic data and compared it to previous approaches and naive tree baselines. We found that TS produces more concise summaries for three out of four baselines, and for the remaining baseline (i.e., Cascading Analysts) that produces summaries of an equivalent size, TS outperforms it in terms of the accuracy of reconstruction.

For future work, we plan to conduct a more detailed study and analysis of summaries involving multiple measures. It is yet unclear what are the advantages and disadvantages of each of the two cost functions, briefly introduced in the previous section. It would be also interesting to see the utility of such multiple measure summaries and whether or not they enable discovery of new data patterns, such as inter-measure correlation.

Bibliography

[1] Deepak Agarwal, Dhiman Barman, Dimitrios Gunopulos, Neal E. Young, Flip Korn, and Divesh Srivastava. Efficient and effective explanation of change in hierarchical summaries. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 6–15, New York, NY, USA, 2007. ACM.

[2] Shaofeng Bu, Laks V. S. Lakshmanan, and Raymond T. Ng. MDL summarization with holes. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 433–444. VLDB Endowment, 2005.

[3] Kareem El Gebaly, Parag Agrawal, Lukasz Golab, Flip Korn, and Divesh Srivastava. Interpretable and informative explanations of outcomes. *Proc. VLDB Endow.*, 8(1):61–72, September 2014.

[4] G. Feng, L. Golab, and D. Srivastava. Scalable informative rule mining. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 437–448, April 2017.

[5] Rob J. Hyndman and Anne B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679 – 688, 2006.

[6] H. V. Jagadish, J. Madar, and Raymond T. Ng. Semantic compression and pattern extraction with fascicles. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 186–198, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[7] Manas Joglekar, Hector Garcia-Molina, and Aditya Parameswaran. Smart drill-down: A new data exploration operator. *Proc. VLDB Endow.*, 8(12):1928–1931, August 2015.

[8] Howard J. Karloff, Flip Korn, Konstantin Makarychev, and Yuval Rabani. On parsimonious explanations for 2-d tree- and linearly-ordered data. In Thomas Schwentick and Christoph Dürr, editors, *28th International Symposium on Theoretical Aspects of Computer Science, STACS 2011, March 10-12, 2011, Dortmund, Germany*, volume 9 of *LIPIcs*, pages 332–343. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[9] Laks V. S. Lakshmanan, Raymond T. Ng, Christine Xing Wang, Xiaodong Zhou, and Theodore J. Johnson. The generalized MDL approach for summarization. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 766–777. VLDB Endowment, 2002.

[10] Laks V. S. Lakshmanan, Jian Pei, and Jiawei Han. Quotient cube: How to summarize the semantics of a data cube. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 778–789. VLDB Endowment, 2002.

[11] Daniel T. Larose. *Discovering Knowledge in Data: An Introduction to Data Mining*. Wiley-Interscience, New York, NY, USA, 2004.

[12] Alberto O. Mendelzon and Ken Q. Pu. Concise descriptions of subsets of structured sets. In *Proceedings of the Twenty-second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '03, pages 123–133, New York, NY, USA, 2003. ACM.

[13] Matthias Ruhl, Mukund Sundararajan, and Qiqi Yan. The cascading analysts algorithm. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 1083–1096, New York, NY, USA, 2018. ACM.

[14] Sunita Sarawagi. Explaining differences in multidimensional aggregates. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 42–53, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[15] Sunita Sarawagi. User-cognizant multidimensional analysis. *The VLDB Journal*, 10(2-3):224–239, September 2001.

[16] Sunita Sarawagi, Rakesh Agrawal, and Nimrod Megiddo. Discovery-driven exploration of OLAP data cubes. In *Proceedings of the 6th International Conference on Extending Database Technology: Advances*

in Database Technology, EDBT '98, pages 168–182, Berlin, Heidelberg, 1998. Springer-Verlag.

[17] Sunita Sarawagi and Gayatri Sathe. I3: Intelligent, interactive investigation of OLAP data cubes. *SIGMOD Rec.*, 29(2):589–, May 2000.

[18] Yannis Sismanis, Antonios Deligiannakis, Nick Roussopoulos, and Yannis Kotidis. Dwarf: Shrinking the petacube. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, SIGMOD '02, pages 464–475, New York, NY, USA, 2002. ACM.

[19] U.S. Census Bureau. Sex by age by educational attainment for the population 18 years and over (B15001), 2016-2017 American community survey 1-year estimates. Data retrieved from American FactFinder, `https://factfinder.census.gov/`.

[20] Jeffrey Scott Vitter and Min Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. *SIGMOD Rec.*, 28(2):193–204, June 1999.

[21] Jeffrey Scott Vitter, Min Wang, and Bala Iyer. Data cube approximation and histograms via wavelets. In *Proceedings of the Seventh International Conference on Information and Knowledge Management*, CIKM '98, pages 96–104, New York, NY, USA, 1998. ACM.

[22] Y. Wen, X. Zhu, S. Roy, and J. Yang. Interactive Summarization and Exploration of Top Aggregate Query Answers. *ArXiv e-prints*, July 2018.