$\mathcal{SPARTAN}$: Using Constrained Models for Guaranteed-Error Semantic Compression

Shivnath Babu[*]
Computer Science Department
Stanford University
Stanford, CA 94305
shivnath@cs.stanford.edu

Minos Garofalakis
Bell Labs, Lucent Technologies
600 Mountain Avenue
Murray Hill, NJ 07974
minos@bell-labs.com

Rajeev Rastogi
Bell Labs, Lucent Technologies
600 Mountain Avenue
Murray Hill, NJ 07974
rastogi@bell-labs.com

ABSTRACT

While a variety of lossy compression schemes have been developed for certain forms of digital data (e.g., images, audio, video), the area of lossy compression techniques for arbitrary data tables has been left relatively unexplored. Nevertheless, such techniques are clearly motivated by the ever-increasing data collection rates of modern enterprises and the need for effective, guaranteed-quality approximate answers to queries over massive relational data sets.

In this paper, we propose $\mathcal{SPARTAN}$, a system that takes advantage of attribute semantics and data-mining models to perform lossy compression of massive data tables. $\mathcal{SPARTAN}$ is based on the novel idea of exploiting predictive data correlations and prescribed error-tolerance constraints for individual attributes to construct concise and accurate *Classification and Regression Tree (CaRT)* models for entire columns of a table. More precisely, $\mathcal{SPARTAN}$ selects a certain subset of attributes (referred to as *predicted* attributes) for which no values are explicitly stored in the compressed table; instead, concise *error-constrained* CaRTs that predict these values (within the prescribed error tolerances) are maintained. To restrict the huge search space of possible CaRT predictors, $\mathcal{SPARTAN}$ uses a Bayesian network structure to guide the selection of CaRT models that minimize the overall storage requirement, based on the prediction and materialization costs for each attribute. $\mathcal{SPARTAN}$'s CaRT-building algorithms employ novel integrated pruning strategies that take advantage of the given error constraints on individual attributes to minimize the computational effort involved. Our experimentation with several real-life data sets offers convincing evidence of the effectiveness of $\mathcal{SPARTAN}$'s model-based approach – $\mathcal{SPARTAN}$ is able to consistently yield substantially better compression ratios than existing semantic or syntactic compression tools (e.g., gzip) while utilizing only small samples of the data for model inference.

1. INTRODUCTION

Effective exploratory analysis of massive, high-dimensional tables of alphanumeric data is a ubiquitous requirement for a variety of application environments, including corporate data warehouses, network traffic monitoring, and large socioeconomic or demographic surveys. For example, large telecommunication providers typically generate and store records of information, termed "Call-Detail Records" (CDRs), for every phone call carried over their network. A typical CDR is a fixed-length record structure comprising several hundred bytes of data that capture information on various (categorical and numerical) attributes of each call; this includes network-level information (e.g., endpoint exchanges), time-stamp information (e.g., call start and end times), and billing information (e.g., applied tariffs), among others [4]. These CDRs are stored in tables that can grow to truly massive sizes, in the order of several TeraBytes per year. Similar massive tables are also generated from network-monitoring tools that gather switch- and router-level traffic data, such as SNMP/RMON probes [19] and Cisco's NetFlow measurement tools [1]. Such tools typically collect traffic information for each network element at fine granularities (e.g., at the level of packet flows between source-destination pairs), giving rise to massive volumes of table data over time. These massive tables of network traffic and CDR data are continuously explored and analyzed to produce the "knowledge" that enables key network-management tasks, including application and user profiling, proactive and reactive resource management, traffic engineering, and capacity planning, as well as providing and verifying Quality-of-Service guarantees for end users.

[*]Work done while visiting Bell Labs.

Traditionally, data compression issues arise naturally in applications dealing with massive data sets, and effective solutions are crucial for optimizing the usage of critical system resources, like storage space and I/O bandwidth (for storing and accessing the data) and network bandwidth (for transferring the data across sites). In mobile-computing applications, for instance, clients are usually disconnected and, therefore, often need to download data for offline use. These clients may use low-bandwidth wireless connections and can be palmtop computers or handheld devices with severe storage constraints. Thus, for efficient data transfer and client-side resource conservation, the relevant data needs to be compressed. Several statistical and dictionary-based compression methods have been proposed for text corpora and multimedia data, some of which (e.g., Lempel-Ziv or Huffman) yield provably optimal asymptotic performance in terms of certain ergodic properties of the data source. These methods, however, fail to provide adequate solutions for compressing a massive data table, as they view the table as a large byte string and do not account for the complex dependency patterns in the table.

Compared to conventional compression methods for text or multimedia data, effectively compressing massive data tables presents a host of novel challenges due to several distinct characteristics of table data sets and their analysis.

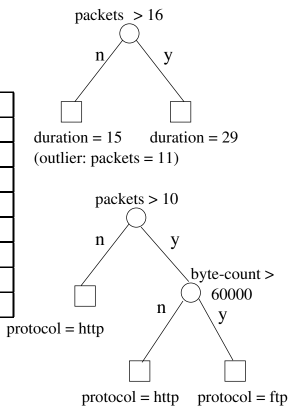
• **Approximate (Lossy) Compression.** Due to the exploratory nature of many data-analysis applications, there are several scenarios in which an exact answer may not be required, and analysts may in fact prefer a fast, approximate answer, as long as the system can guarantee that *user-prescribed constraints on the approximation error* are met. For example, during a drill-down query sequence in ad-hoc data mining, initial queries in the sequence frequently have the sole purpose of determining the truly interesting queries and regions of the data table. Providing (reasonably accurate) approximate answers to these initial queries gives analysts the ability to focus their explorations quickly and effectively, without consuming inordinate amounts of valuable system resources. Thus, in contrast to traditional lossless data compression, the compression of massive tables can often afford to be *lossy*, as long as some (user- or application-defined) upper bounds on the compression error are guaranteed by the compression algorithm. This is obviously a crucial differentiation, as even small error tolerances can help us achieve much better compression ratios.

• **Semantic Compression.** Existing compression techniques are "syntactic" in the sense that they operate at the level of consecutive bytes of data. As explained above, such syntactic methods typically fail to provide adequate solutions for table-data compression, since they essentially view the data as a large byte string and do not exploit the complex dependency patterns in the table. Effective table compression mandates techniques that are *semantic* in nature, in the sense that they account for and exploit both (1) the meanings and dynamic ranges of individual attributes (e.g., by taking advantage of the specified error tolerances); and, (2) existing data dependencies and correlations among attributes in the table.

In this paper, we describe the architecture of $\mathcal{SPARTAN}$[1], a system that takes advantage of attribute semantics and data-mining models to perform lossy compression of massive data tables [2]. $\mathcal{SPARTAN}$ is based on the novel idea of exploiting data correlations and user-specified "loss"/error tolerances for individual attributes to construct concise and accurate *Classification and Regression Tree (CaRT)* models [3] for entire columns of a table. More precisely, $\mathcal{SPARTAN}$ selects a certain subset of attributes (referred to as *predicted* attributes) for which no values are explicitly stored in the compressed table; instead, concise CaRTs that predict these values (within the prescribed error bounds) are maintained. Thus, for a predicted attribute X that is strongly correlated with other attributes in the table, $\mathcal{SPARTAN}$ is typically able to obtain a very succinct CaRT predictor for the values of X, which can then be used to completely eliminate the column for X in the compressed table. Clearly, storing a compact CaRT model in lieu of millions or billions of actual attribute values can result in substantial savings in storage. In addition, allowing for errors in the attribute values predicted by a CaRT model only serves to reduce the size of the model even further and, thus, improve the quality of compression; this is because, as is well known, the size of a CaRT model is typically inversely correlated to the accuracy with which it models a given set of values [3; 16].

protocol	duration	byte-count	packets
http	12	2,000	1
http	16	24,000	5
ftp	27	100,000	24
http	15	20,000	8
ftp	32	300,000	35
http	19	40,000	11
http	26	58,000	18
ftp	18	80,000	15

(a) Tuples in Table



(b) CaRT Models

Figure 1: Model-Based Semantic Compression.

EXAMPLE 1.1.: *Consider the table with 4 attributes and 8 tuples shown in Figure 1(a), where each tuple represents a* data flow *in an IP network. The categorical attribute* protocol *records the application-level protocol generating the flow; the numeric attribute* duration *is the time duration of the flow; and, the numeric attributes* byte-count *and* packets *capture the total number of bytes and packets transferred. Let the acceptable errors due to compression for the numeric attributes* duration, byte-count, *and* packets *be 3, 1,000, and 1, respectively. Also, assume that the* protocol *attribute has to be compressed without error (i.e., zero tolerance). Figure 1(b) depicts a regression tree for predicting the* duration *attribute (with* packets *as the predictor attribute) and a classification tree for predicting the* protocol *attribute (with* packets *and* byte-count *as the predictor attributes). Observe that in the regression tree, the predicted value of duration (label value at each leaf) is almost always within 3, the specified error tolerance, of the actual tuple value. For instance, the predicted value of* duration *for the tuple with* packets = *1 is 15 while the original value is 12. The only tuple for which the predicted value violates this error bound is the tuple with* packets = *11, which is an marked as an outlier value in the regression tree. There are no outliers in the classification tree. By explicitly storing, in the compressed version of the table, each outlier value along with the CaRT models and the projection of the table onto only the predictor attributes (*packets *and* byte-count*), we can ensure that the error due to compression does not exceed the user-specified bounds. Further, storing the CaRT models (plus outliers) for* duration *and* protocol *instead of the attribute values themselves results in a reduction from 8 to 4 values for* duration *(2 labels for leaves + 1 split value at internal node + 1 outlier) and a reduction from 8 to 5 values for* protocol *(3 labels for leaves + 2 split values at internal nodes).* ∎

The key algorithmic problem faced by $\mathcal{SPARTAN}$'s compression engine is that of computing an optimal set of CaRT models for the input table such that (a) the overall storage requirements of the compressed table are minimized, and (b) all predicted attribute values are within the user-specified error constraints. This is a very challenging optimization problem since, not only is there an exponential number of possible CaRT-based models to choose from, but also building CaRTs (to estimate their compression benefits) is a computation-intensive task, typically requiring multiple passes over the data [3; 12; 18]. As a consequence, $\mathcal{SPARTAN}$ has to employ a number of sophisticated techniques from the areas of knowledge discovery and combinatorial optimization in order to efficiently discover a "good" (sub)set of predicted attributes and construct the corresponding CaRT models. Below, we list some of $\mathcal{SPARTAN}$'s salient features.

- **Use of Bayesian Network to Uncover Data Dependencies.** A Bayesian network is a DAG whose edges reflect strong predictive correlations between nodes of the graph [14; 15]. Thus, a Bayesian network on the table's attributes can be used to dramatically reduce the search space of potential CaRT models since, for any attribute, the most promising CaRT predictors are the ones that involve attributes in its "neighborhood" in the network. Our current $\mathcal{SPARTAN}$ implementation uses a constraint-based Bayesian network builder based on recently proposed algorithms for efficiently inferring Bayesian structure from data. To control the computa-

[1] [From Webster] **Spartan:** */'spart-*n/* (1) of or relating to Sparta in ancient Greece, (2) a: marked by strict self-discipline and avoidance of comfort and luxury, b: sparing of words : TERSE : LACONIC.

tional overhead, the Bayesian network is built using a reasonably small random sample of the input table.

- **Novel CaRT-selection Algorithms that Minimize Storage Cost.** $\mathcal{SPARTAN}$ exploits the inferred Bayesian network structure by using it to intelligently guide the selection of CaRT models that minimize the overall storage requirement, based on the prediction and materialization costs for each attribute. Intuitively, the goal is to minimize the sum of the prediction costs (for predicted attributes) and materialization costs (for attributes used in the CaRTs). We demonstrate that this model-selection problem is a strict generalization of the *Weighted Maximum Independent Set (WMIS)* problem [6; 10], which is known to be $\mathcal{NP}$-hard. However, by employing a novel algorithm that effectively exploits the discovered Bayesian structure in conjunction with efficient, near-optimal WMIS heuristics, $\mathcal{SPARTAN}$ is able to obtain a good set of CaRT models for compressing the table.

- **Improved CaRT Construction Algorithms that Exploit Error Constraints.** A signification portion of $\mathcal{SPARTAN}$'s execution time is spent in building CaRT models. This is mainly because $\mathcal{SPARTAN}$ needs to actually construct many promising CaRTs in order to estimate their prediction cost, and CaRT construction is a computationally-intensive process. To reduce CaRT-building times and speed up system performance, $\mathcal{SPARTAN}$ employs the following three optimizations: (1) CaRTs are built using random samples instead of the entire data set, (2) leaves are not expanded if values of tuples in them can be predicted with acceptable accuracy, and (3) pruning is integrated into the tree growing phase using novel algorithms that exploit the prescribed error-tolerance constraints for the predicted attribute.

We have implemented the $\mathcal{SPARTAN}$ system and conducted an extensive experimental study with three real-life data sets to compare the quality of compression due to $\mathcal{SPARTAN}$'s model-based approach with existing syntactic and semantic compression techniques. For all three data sets, and even for small error tolerances (e.g., 1%), we found that $\mathcal{SPARTAN}$ is able to achieve, on an average, 20-30% better compression ratios. Further, our experimental results indicate that $\mathcal{SPARTAN}$ compresses tables better when they contain more numeric attributes and as error thresholds grow bigger. For instance, for a table containing mostly numeric attributes and for higher error tolerances in the 5-10% range, $\mathcal{SPARTAN}$ outperformed existing compression techniques by more than a factor of 3. Finally, we show that our improved CaRT construction algorithms make $\mathcal{SPARTAN}$'s performance competitive, enabling it to compress data sets containing more than half a million tuples in a few minutes. Thus, our experimental results clearly demonstrate the effectiveness of $\mathcal{SPARTAN}$'s methodology for compressing massive tables.

The technical discussion in this overview paper will focus mostly on $\mathcal{SPARTAN}$'s efficient CaRT construction algorithms and how they exploit the prescribed error constraints for predicted attributes. For more details on other $\mathcal{SPARTAN}$ components the interested reader is referred to [2].

2. PROBLEM FORMULATION AND OVERVIEW OF OUR APPROACH

In this section, we describe our proposed *model-based* framework for the semantic compression of massive data tables and we provide an overview of the architecture of $\mathcal{SPARTAN}$, a system built around this framework. We start by providing some necessary definitions and background information on compression and data mining models.

2.1 Preliminaries

Definitions and Notation. The input to the $\mathcal{SPARTAN}$ system consists of a n-attribute table T, comprising a large number of tuples (rows). We let $\mathcal{X} = \{X_1, \ldots, X_n\}$ denote the set of n attributes of T and $dom(X_i)$ represent the domain of attribute X_i. Attributes with a discrete, unordered value domain are referred to as *categorical*, whereas those with ordered value domains are referred to as *numeric*. We also use T_c to denote the compressed version of table T, and $|T|$ ($|T_c|$) to denote the storage-space requirements for T (T_c) in bytes.

The key input parameter to our semantic compression algorithms is a (user- or application-specified) n-dimensional vector of *error tolerances* $\bar{e} = [e_1, \ldots, e_n]$ that defines the *per-attribute* acceptable degree of information loss when compressing T. (Per-attribute error constraints are also employed in the fascicles framework [11].) Intuitively, the i^{th} entry of the tolerance vector e_i specifies an upper bound on the error by which any (approximate) value of X_i in the compressed table T_c can differ from its original value in T. Our error tolerance semantics differ across categorical and numeric attributes, due to the very different nature of the two attribute classes.

1. *For a numeric attribute* X_i, the tolerance e_i defines an upper bound on the *absolute difference* between the actual value of X_i in T and the corresponding (approximate) value in the compressed table T_c. That is, if x, x' denote the accurate and approximate value (respectively) of attribute X_i for *any* tuple of T, then our compressor guarantees that $x \in [x' - e_i, x' + e_i]$.

2. *For a categorical attribute* X_i, the tolerance e_i defines an upper bound on the *probability* that the (approximate) value of X_i in T_c is different from the actual value in T. More formally, if x, x' denote the accurate and approximate value (respectively) of attribute X_i for *any* tuple of T, then our compressor guarantees that $P[x = x'] \geq 1 - e_i$.

For numeric attributes, the error tolerance could very well be specified in terms of quantiles of the overall range of values rather than absolute, constant values. Similarly, for categorical attributes the probability of error could be specified separately for each individual attribute class (i.e., value) rather than an overall measure. (Note that such an extension would, in a sense, make the error bounds for categorical attributes more "local", similar to the numeric case.) Our proposed model-based compression framework and algorithms can be readily extended to handle these scenarios, so the specific definitions of error tolerance are not central to our methodology. To make our discussion concrete, we use the definitions outlined above for the two attribute classes. (Note that our error-tolerance semantics can also easily capture *lossless* compression as a special case, by setting $e_i = 0$ for all i.)

Metrics. The basic metric used to compare the performance of different compression algoritms is the well-known *compression ratio*, defined as the ratio of the size of the compressed data representation produced by the algorithm and the size of the original (uncompressed) input. A secondary performance metric is the *compression throughput* that, intuitively, corresponds to the rate at which a compression algorithm can process data from its input; this is typically defined as the size of the uncompressed input divided by the total compression time.

Our work focuses primarily on optimizing the compression ratio, that is, achieving the maximum possible reduction in the size of the data within the acceptable levels of error defined by the user. This choice is mainly driven by the massive, long-lived data sets that are characteristic of our target data warehousing applications and the observation that the computational cost of effective compression can be amortized over the numerous physical operations (e.g., transmissions over a low-bandwidth link) that will take place during the lifetime of the data. Also, note that our methodology offers a key "knob" for tuning compression throughput performance, namely the size of the data sample used by $\mathcal{SPARTAN}$'s model-construction algorithms. Setting the sample size based on the amount of main memory available in the system can help ensure high compression speeds.

2.2 Model-Based Semantic Compression: Problem Statement

Briefly, our proposed *model-based* framework for the semantic compression of tables is based on two key technical ideas. First, we exploit the (user- or application-specified) error constraints on individual attributes in conjunction with data mining techniques to efficiently build *accurate models* of the data. Second, we compress the input table using a select subset of the models built. The basic intuition here is that this select subset of data-mining models is carefully chosen to capture large portions of the input table within the specified error bounds.

More formally, we define the model-based, compressed version of the input table T as a pair $T_c =< T', \{\mathcal{M}_1, \dots, \mathcal{M}_p\} >$ where (1) T' is a small (possibly empty) projection of the data values in T that are retained *accurately* in T_c; and, (2) $\{\mathcal{M}_1, \dots, \mathcal{M}_p\}$ is a select set of data-mining models, carefully built with the purpose of maximizing the degree of compression achieved for T while obeying the specified error-tolerance constraints. Abstractly, the role of the projection T' is to capture values (tuples or sub-tuples) of the original table that cannot be effectively "summarized away" in a compact data-mining model within the specified error tolerances. (Some of these values may in fact be needed as *input* to the selected models.) The attribute values in T' can either be retained as uncompressed data or be compressed using a conventional lossless algorithm.

A definition of our general model-based semantic compression problem can now be stated as follows.

[Model-Based Semantic Compression (MBSC)] Given a massive, multi-attribute table T and a vector of (per-attribute) error tolerances $\bar{e}$, find a collection of models $\{\mathcal{M}_1, \dots, \mathcal{M}_m\}$ and a compression scheme for T based on these models $T_c =< T', \{\mathcal{M}_1, \dots, \mathcal{M}_p\} >$ such that the specified error bounds $\bar{e}$ are not exceeded and the storage requirements $|T_c|$ of the compressed table are minimized. ∎

Given the multitude of possible models that can be extracted from the data, this is obviously a very general problem definition that covers a huge design space of possible alternatives for semantic compression. We provide a more concrete statement of the problem addressed in our work on the $\mathcal{SPARTAN}$ system later in this section. First, however, we discuss how our model-based compression framework relates to recent work on semantic compression and demonstrate the need for the more general approach advocated in this paper.

Comparison with Fascicles. Our model-based semantic compression framework, in fact, generalizes earlier ideas for semantic data compression, such as the very recent proposal of Jagadish, Madar, and Ng on using *fascicles* for the semantic compression of relational tables [11]. (To the best of our knowledge, this is the only work on lossy semantic compression of tables with guaranteed upper bounds on the compression error.)

A fascicle basically represents a collection of tuples (rows) that have *approximately* matching values for some (but not necessarily all) attributes, where the degree of approximation is specified by user-provided compactness parameters. Essentially, fascicles can be seen as a specific form of data-mining models, i.e., clusters in subspaces of the full attribute space, where the notion of a cluster is based on the acceptable degree of loss during data compression. The key idea of fascicle-based semantic compression is to exploit the given error bounds to allow for aggressive grouping and "summarization" of values by clustering multiple rows of the table along several columns (i.e., the dimensionality of the cluster).

EXAMPLE 2.1.: *Consider the table in Figure 1(a) described in Example 1.1. Error tolerances of 3, 1,000 and 1 for the three numeric attributes* duration*,* byte-count *and* packets*, respectively, result in the following two fascicles (among others):*

F_1

http	**12**	*2,000*	*1*
http	**15**	*20,000*	*8*
http	**16**	*24,000*	*5*

F_2

ftp	**27**	*100,000*	*24*
ftp	**32**	*300,000*	*35*

The tuples in the two fascicles F_1 and F_2 are similar (with respect to the permissible errors) on the protocol *and* duration *attributes. (Two numeric attribute values are considered similar if the difference between them is at most twice the error bound for that attribute.) Substituting for each numeric attribute value, the mean of the maximum and minimum value of the attribute in a fascicle ensures that the introduced error is acceptable. Consequently, in order to compress the table using fascicles, the single (sub)tuple (http, 14) replaces the three corresponding (sub)tuples in the first fascicle and (ftp, 29.5)replaces the two subtuples in the second fascicle. Thus, in the final compressed table, the maximum error for* duration *is not greater than 3, and the number of values stored for the* protocol *and* duration *attributes is reduced from 8 to 5.* ∎

As the above example shows, in many practical cases, fascicles can effectively exploit the specified error tolerances to achieve high compression ratios. There are however, several scenarios for which a more general, model-based compression approach is in order. The main observation here is that fascicles only try to detect "row-wise" patterns, where sets of rows have similar values for several attributes. Such "row-wise" patterns within the given error-bounds can be impossible to find when strong "column-wise" patterns/dependencies (e.g., functional dependencies) exist across attributes of the table. On the other hand, data-mining models like CaRTs capture and model attribute correlations and, thereby, can attain much better semantic compression when such correlations exist. Revisiting Example 1.1, we see that CaRTs result in better compression than fascicles even for our toy example table[2] – the storage for the *duration* attribute reduces from 8 to 4 with CaRTs compared to 5 with fascicles.

Concrete Problem Definition. The above discussion demonstrates the need for a semantic compression methodology that is more general than simple fascicle-based row clustering in that it can account for and exploit strong dependencies among the attributes of the input table. The important observation here (already outlined in

[2]This is not surprising given the strong correlations among the attributes in a table of network traffic records [1].

Example 1.1) is that data mining offers models (i.e., CaRTs) that can accurately capture such dependencies with very concise data structures. Thus, in contrast to fascicles, our general model-based semantic compression paradigm can accommodate such scenarios. The ideas of row-wise pattern discovery and clustering for semantic compression have been thoroughly explored in the context of fascicles [11]. In contrast, our work on the $\mathcal{SPARTAN}$ semantic compressor reported in this paper focuses primarily on the novel problems arising from the need to effectively detect and exploit (column-wise) attribute dependencies for the purposes of semantic table compression. The key idea underlying our approach is that, in many cases, a small classification (regression) tree structure can be used to accurately *predict* the values of a categorical (resp., numeric) attribute (based on the values of other attributes) for a very large fraction of table rows. This means that, for such cases, our compression algorithms can completely *eliminate* the predicted column in favor of a compact *predictor* (i.e., a classification or regression tree model) and a small set of outlier column values. More formally, the design and architecture of $\mathcal{SPARTAN}$ focuses mainly on the following concrete MBSC problem.

[$\mathcal{SPARTAN}$ **CaRT-Based Semantic Compression**] Given a massive, multi-attribute table T with a set of categorical and/or numeric attributes $\mathcal{X}$, and a vector of (per-attribute) error tolerances $\bar{e}$, find a subset $\{X_1, \ldots, X_p\}$ of $\mathcal{X}$ and a collection of corresponding CaRT models $\{\mathcal{M}_1, \ldots, \mathcal{M}_p\}$ such that: (1) model $\mathcal{M}_i$ is a predictor for the values of attribute X_i based solely on attributes in $\mathcal{X} - \{X_1, \ldots, X_p\}$, for each $i = 1, \ldots, p$; (2) the specified error bounds $\bar{e}$ are not exceeded; and, (3) the storage requirements $|T_c|$ of the compressed table $T_c =< T', \{\mathcal{M}_1, \ldots, \mathcal{M}_p\} >$ are minimized. ∎

Abstractly, our novel semantic compression algorithms seek to partition the set of input attributes $\mathcal{X}$ into a set of *predicted attributes* $\{X_1, \ldots, X_p\}$ and a set of *predictor attributes* $\mathcal{X} - \{X_1, \ldots, X_p\}$ such that the values of each predicted attribute can be obtained within the specified error bounds based on (a subset of) the predictor attributes through a small classification or regression tree (except perhaps for a small set of outlier values). (We use the notation $\mathcal{X}_i \longrightarrow X_i$ to denote a CaRT predictor for attribute X_i using the set of predictor attributes $\mathcal{X}_i \subseteq \mathcal{X} - \{X_1, \ldots, X_p\}$.) Note that we do not allow a predicted attribute X_i to also be a predictor for a different attribute. This restriction is important since predicted values of X_i can contain errors, and these errors can cascade further if the erroneous predicted values are used as predictors, ultimately causing error constraints to be violated. The final goal, of course, is to minimize the overall storage cost of the compressed table. This storage cost $|T_c|$ is the sum of two basic components:

1. *Materialization cost*, i.e., the cost of storing the values for all predictor attributes $\mathcal{X} - \{X_1, \ldots, X_p\}$. This cost is represented in the T' component of the compressed table, which is basically the projection of T onto the set of predictor attributes. (The storage cost of materializing attribute X_i is denoted by $\texttt{MaterCost}(X_i)$.)

2. *Prediction cost*, i.e., the cost of storing the CaRT models used for prediction plus (possibly) a small set of outlier values of the predicted attribute for each model. (The storage cost of predicting attribute X_i through the CaRT predictor $\mathcal{X}_i \longrightarrow X_i$ is denoted by $\texttt{PredCost}(\mathcal{X}_i \longrightarrow X_i)$; note that this does *not* include the cost of materializing the predictor attributes in $\mathcal{X}_i$.)

We should note here that our proposed CaRT-based compression methodology is essentially *orthogonal* to techniques based on row-wise clustering, like fascicles. It is entirely possible to combine the two techniques for an even more effective model-based semantic compression mechanism. As an example, the predictor attribute table T' derived by our "column-wise" techniques can be compressed using a fascicle-based algorithm. (In fact, this is exactly the strategy used in our current $\mathcal{SPARTAN}$ implementation; however, other methods for combining the two are also possible.) The important point here is that, since the entries of T' are used as inputs to (approximate) CaRT models for other attributes, care must be taken to ensure that errors introduced in the compression of T' do not compound over the CaRT models in a way that causes error guarantees to be violated. More details can be found in [2].

2.3 Overview of the $\mathcal{SPARTAN}$ System

As depicted in Figure 2, the architecture of the $\mathcal{SPARTAN}$ system comprises of four major components: the DependencyFinder, the CaRTSelector, the CaRTBuilder, and the RowAggregator. In the following, we provide a brief overview of each $\mathcal{SPARTAN}$ component.

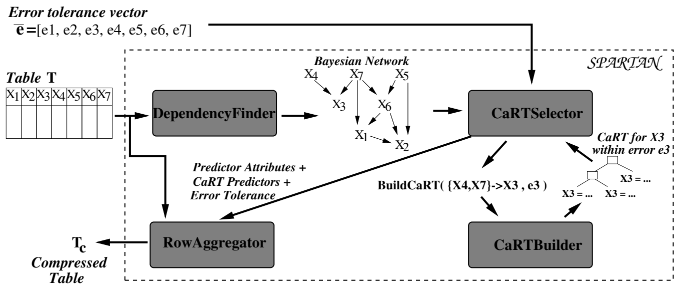


Figure 2: $\mathcal{SPARTAN}$ system architecure.

• <u>DependencyFinder.</u> The purpose of the DependencyFinder component is to produce an *interaction model* for the input table attributes, that is then used to guide the CaRT building algorithms of $\mathcal{SPARTAN}$. The main observation here is that, since there is an exponential number of possibilities for building CaRT-based attribute predictors, we need a concise model that identifies the strongest correlations and "predictive" relationships in the input data.

The approach used in the DependencyFinder component of $\mathcal{SPARTAN}$ is to construct a *Bayesian network* [14] on the underlying set of attributes $\mathcal{X}$. Abstractly, a Bayesian network imposes a Directed Acyclic Graph (DAG) structure G on the set of nodes $\mathcal{X}$, such that directed edges capture direct statistical dependence between attributes. In effect, a Bayesian network is a graphical specification of a joint probability distribution that is believed to have generated the observed data. Bayesian networks are an essential tool for capturing causal and/or predictive correlations in observational data; such interpretations are typically based on well-known dependence semantics of the Bayesian network structure [14; 15]. Intuitively, a set of nodes in the "neighborhood" of node X_i in G (e.g., X_i's parents) captures the attributes that are strongly correlated to X_i and, therefore, show promise as possible predictor attributes for X_i [2].

• <u>CaRTSelector.</u> The CaRTSelector component constitutes the core of $\mathcal{SPARTAN}$'s model-based semantic compression engine. Given the input table T and error tolerances e_i, as well as the Bayesian network on the attributes of T built by the DEPENDENCYFINDER, the CARTSELECTOR is responsible for selecting a collection of predicted attributes and the corresponding CaRT-based predictors such that the final overall storage cost is minimized (within the given error bounds). As discussed above, $\mathcal{SPARTAN}$'s CARTSELECTOR employs the Bayesian network G built on $\mathcal{X}$ to intelligently guide the search through the huge space of possible attribute prediction strategies. Clearly, this search involves repeated interactions with the CARTBUILDER component, which is responsible for actually building the CaRT-models for the predictors (Figure 2).

We demonstrate that even in the simple case where the set of nodes that is used to predict an attribute node in G is *fixed*, the problem of selecting a set of predictors that minimizes the combination of materialization and prediction cost naturally maps to the *Weighted Maximum Independent Set (WMIS)* problem, which is known to be $\mathcal{NP}$-hard and notoriously difficult to approximate [6; 10]. Based on this observation, we propose a CaRT-model selection strategy that starts out with an initial solution obtained from a near-optimal heuristic for WMIS [9; 10] and tries to incrementally improve it by small perturbations based on the unique characteristics of our problem. We also give an alternative *greedy* model-selection algorithm that chooses its set of predictors using a simple local condition during a single "roots-to-leaves" traversal of the Bayesian network G.

• CARTBUILDER. Given a collection of predicted and (corresponding) predictor attributes $\mathcal{X}_i \longrightarrow X_i$, the goal of the CARTBUILDER component is to efficiently construct CaRT-based models for each X_i in terms of $\mathcal{X}_i$ for the purposes of semantic compression. Induction of CaRT-based models is typically a computation-intensive process that requires multiple passes over the input data [3; 18; 12]. As we demonstrate, however, $\mathcal{SPARTAN}$'s CaRT-construction algorithms can take advantage of the compression semantics and exploit the user-defined error-tolerance constraints to effectively prune computation. In addition, by building CaRTs using data samples instead of the entire data set, $\mathcal{SPARTAN}$ is able to further speed up model construction.

• ROWAGGREGATOR. Once $\mathcal{SPARTAN}$'s CARTSELECTOR component has finalized a "good" solution to the CaRT-based semantic compression problem, it hands off its solution to the ROWAGGREGATOR component which tries to further improve the compression ratio through row-wise clustering. Briefly, the ROWAGGREGATOR uses a fascicle-based algorithm [11] to compress the predictor attributes, while ensuring (based on the CaRT models built) that errors in the predictor attribute values are not propagated through the CaRTs in a way that causes error tolerances (for predicted attributes) to be exceeded.

The next section discusses $\mathcal{SPARTAN}$'s CARTBUILDER component in detail, focusing on our novel CaRT-construction algorithms that effectively exploit user-prescribed error constraints on individual attributes. Details of the other three $\mathcal{SPARTAN}$ components can be found in the original $\mathcal{SPARTAN}$ paper [2].

3. THE $\mathcal{SPARTAN}$ CARTBUILDER

The CARTBUILDER component of $\mathcal{SPARTAN}$ constructs a CaRT predictor $\mathcal{X}_i \longrightarrow X_i$ for the attribute X_i with $\mathcal{X}_i$ as the predictor attributes. The CARTBUILDER component's objective is to construct the smallest (in terms of storage space) CaRT model such that each predicted value (of a tuple's value for attribute X_i) deviates from the actual value by at most e_i, the prescribed error tolerance for attribute X_i.

If the predicted attribute X_i is categorical, then the CARTBUILDER component builds a compact classification tree with values of X_i serving as class labels. CARTBUILDER employs classification tree construction algorithms from [18; 17] to first construct a low storage cost tree and then explicitly stores sufficient number of outliers such that the fraction of misclassified records is less than the specified error bound e_i. Thus, CARTBUILDER guarantees that the fraction of attribute X_i's values that are incorrectly predicted is less than e_i.

In the remainder of this section, we focus on the case when the predicted attribute X_i is numeric. Specifically, we present efficient algorithms for constructing compact regression trees for predicting X_i with an error that does not exceed e_i.

3.1 Storage Cost of Regression Trees

A regression tree consists of two types of nodes – internal nodes and leaves. Each internal node is labeled with a splitting condition involving attribute $X_j \in \mathcal{X}_i$ – this condition is of the form $X_j > x$ if X_j is a numeric attribute and $X_j \in \{x, x', \dots\}$ if X_j is categorical. Each leaf is labeled with a numeric value x which is the predicted value for X_i for all tuples in table T that belong to the leaf (a tuple belongs to a leaf if it satisfies the sequence of splitting conditions on the path from the root to the leaf). Thus, for a tuple t belonging to a leaf with label x, the predicted value of t on attribute X_i, $t[X_i]$, satisfies the error bounds if $t[X_i] \in [x - e_i, x + e_i]$. Tuples t in the leaf for whom $t[X_i]$ lies outside the range $[x - e_i, x + e_i]$ are *outliers* since their predicted values differ from their actual values by more than the tolerance limit.

The storage cost of a regression tree R for predicting X_i thus comprises (1) the cost of encoding nodes of the tree and their associated labels, and (2) the cost of encoding outliers. The cost of encoding an internal node N of the tree is $1 + \log|\mathcal{X}_i| + C_{split}(N)$, where 1 bit is needed to specify the type of node (internal or leaf), $\log|\mathcal{X}_i|$ is the number of bits to specify the splitting attribute and $C_{split}(N)$ is the cost of encoding the split value for node N. If v is the number of distinct values for the splitting attribute X_j at node N, then $C_{split}(N) = \log(v - 1)$ if X_j is numeric and $C_{split}(N) = \log(2^v - 2)$ if X_j is categorical. Next, we compute the cost of encoding a leaf with label x. Due to Shannon's theorem, in general, the number of bits required to store m values of attribute X_i is m times the entropy of X_i. Since X_i is a numeric attribute, $\log|dom(X_i)|$ is a good approximation for the entropy of X_i. Thus, to encode a leaf node N, we need $1 + \log(|dom(X_i)|)$ bits, where 1 bit is needed to encode the node type for the leaf and $\log(|dom(X_i)|)$ bits are used to encode the label. Finally, if the leaf contains m outliers, then these need to be encoded separately at a total cost of approximately $m \log(|dom(X_i)|)$.

In the following subsections, we present efficient algorithms for computing a low-cost regression tree that predicts X_i.

3.2 Regression Tree Construction with Separate Building and Pruning

We construct a low-cost regression tree in two phases – a tree building phase followed by a tree pruning phase. At the start of the building phase, the tree contains a single root node containing all the tuples in T. The tree building procedure continues to split each leaf N in the tree until for tuples t in the leaf, the difference between the maximum and minimum values of $t[X_i]$ is less than or equal to $2e_i$. The splitting condition for a node N containing a set of tuples S is chosen such that the mean square error of the two sets of tuples due to the split is minimized. Thus, if the split partitions S into two sets of tuples S_1 and S_2, then $\sum_{t \in S_1}(t[X_i] - \mu_1)^2 + \sum_{t \in S_2}(t[X_i] - \mu_2)^2$ is minimized, where

μ_1 and μ_2 are the means of $t[X_i]$ for tuples t in S_1 and S_2, respectively.

At the end of the tree building phase, for each leaf N of the constructed tree R, the label x is set to $(t_{min}[X_i] + t_{max}[X_i])/2$, where t_{min} and t_{max} are the tuples in N with the minimum and maximum values for X_i. Thus, R contains no outliers since $(t_{max}[X_i] - t_{min}[X_i]) \leq 2e_i$ and as a result, the error in the predicted values of attribute X_i are within the permissible limit. However, the cost of R may not be minimum – specifically, deleting nodes from R may actually result in a tree with smaller storage cost. This is because while pruning an entire subtree from R may introduce outliers whose values need to be stored explicitly, the cost of explicitly encoding outliers may be much smaller than the cost of the deleted subtree.

Thus, the goal of the pruning phase is to find the subtree of R (with the same root as R) with the minimum cost. Consider an internal node N in R and let S be the set of tuples in N. Let R_N be the subtree of R rooted at node N with cost $C(R_N)$ (the cost of encoding nodes and outliers in R_N). Thus, $C(R_N)$ is essentially the reduction in the cost of R if N's children are deleted from R. Now, deletion of N's children from R causes N to become a leaf whose new cost is as follows. Suppose that x is the label value for N that minimizes the number, say m, of outliers. Then the new cost of leaf N, $C(N) = 1 + \log(|dom(X_i)|) + m\log(|dom(X_i)|)$. Thus, if $C(N) \leq C(R_N)$ for node N, then deleting N's children from R causes R's cost to decrease.

The overall pruning algorithm for computing the minimum cost subtree of R considers the nodes N in R in decreasing order of their distance from the root of R. If, for a node N, $C(N) \leq C(R_N)$, then its children are deleted from R. One issue that we still need to resolve when computing $C(N)$ for a node N is determining the label x for N that minimizes the number, m, of outliers. This can easily be achieved by maintaining for each node N, a sorted list containing the X_i values of tuples in N. Then, in a single pass over the list, for each value x' in the list, it is possible to compute the number of elements (in the list) that fall in the window $[x', x'+2e_i]$. If x' is the value for which the window $[x', x'+2e_i]$ contains the maximum number of elements, then the label x for node N is set to $x' + e_i$ (since this would minimize the number of outliers).

3.3 Regression Tree Construction with Integrated Building and Pruning

In the tree construction algorithm presented in the previous subsection, portions of tree R are pruned only after R is completely built. Consequently, the algorithm may expend substantial effort on building portions of the tree that are subsequently pruned. In this subsection, we present an algorithm that during the growing phase, first determines if a node will be pruned during the following pruning phase, and subsequently stops expanding such nodes. Thus, integrating the pruning phase into the building phase enables the algorithm to reduce the number of expanded tree nodes and improve performance. Although Rastogi and Shim [17] present integrated algorithms for classification trees, the algorithms we present in this subsection are novel since in our case, we are primarily interested in regression trees and we allow bounded errors in predicted values.

Recall that for a completely built regression tree R, for a non-leaf node N in R, we pruned N's children if $C(N) \leq C(R_N)$, where $C(R_N)$ and $C(N)$ are the costs of encoding the subtree R_N and node N (considering it to be a leaf), respectively. However, if R is a partially built regression tree, then R_N may still contain some leaves that are eligible for expansion. As a result, $C(R_N)$, the cost of the partial subtree R_N, may be greater than the cost of the fully

```
procedure LowerBound($N$, $e_i$, $b$)
Input: Leaf $N$ for which lower bound on subtree cost is to be computed;
       error tolerance $e_i$ for attribute $X_i$; bound $b$ on the maximum number
       of internal nodes in subtree rooted at $N$.
Output: Lower bound $L(N)$ on cost of subtree rooted at $N$.
begin
1.  for $i := 1$ to $r$
2.     minOut$[i, 0] := i$
3.  for $j := 1$ to $b+1$
4.     minOut$[0, j] := 0$
5.  $l := 0$
6.  for $i := 1$ to $r$
7.     while $x_i - x_{l+1} > 2e_i$
8.        $l := l + 1$
9.     for $j := 1$ to $b+1$
10.       minOut$[i, j] := \min\{$minOut$[i-1, j] + 1$, minOut$[l, j-1]\}$
11. end
12. $L(N) := \infty$
13. for $j := 0$ to $b$
14.    $L(N) := \min\{\ L(N)\ ,\ 2j + 1 + j\log(|\mathcal{X}_i|) +$
                  $(j + 1 +$ minOut$(r, j+1))\log(|dom(X_i)|)\ \}$
15. $L(N) := \min\{\ L(N)\ ,\ 2b + 3 +$
                  $(b+1)\log(|\mathcal{X}_i|) + (b+2)\log(|dom(X_i)|)\ \}$
16. return $L(N)$
end
```

Figure 3: Algorithm for Estimating Lower Bound on Subtree Cost.

expanded subtree rooted at N (after "still to be expanded" leaves in R_N are completely expanded). This overestimation by $C(R_N)$ of the cost of the fully expanded subtree rooted at N can result in N's children being wrongly pruned (assuming that we prune N's children if $C(N) \leq C(R_N)$).

Instead, suppose that for a "still to be expanded leaf" N, we could compute $L(N)$, a lower bound on the cost of any fully expanded subtree rooted at N. Further, suppose for a non-leaf node N, we define $L(R_N)$ to be the sum of (1) for each internal node N' in R_N, $1 + \log(|\mathcal{X}_i|) + C_{split}(N')$, (2) for each "still to be expanded" leaf node N' in R_N, $L(N')$ and (3) for leaf nodes N' in R_N that do not need to be expanded further and containing m outliers, $1 + (m + 1)\log(|dom(X_i)|)$. It is relatively straightforward to observe that $L(R_N)$ is indeed a lower bound on the cost of any fully expanded subtree rooted at node N. As a consequence, if $C(N) \leq L(R_N)$, then we can safely prune N's children from R since $C(N)$ would be less than or equal to the cost of the fully expanded subtree rooted at N and as a result, N's children would be pruned from R during the pruning phase anyway.

Thus, we simply need to be able to estimate a lower bound $L(N)$ on the cost of any fully expanded subtree rooted at a "still to be expanded" leaf N. A simple estimate for the lower bound $L(N)$ is $1 + \min\{\log(|\mathcal{X}_i|), \log(|dom(X_i)|)\}$. However, in the following, we show how a better estimate for $L(N)$ can be devised. Let $x_1, x_2, \ldots, x_r$ be the values of attribute X_i for tuples in node N in sorted order. Suppose we are permitted to use k intervals of width $2e_i$ to cover values in the sorted list. Further, suppose we are interested in choosing the intervals such that the number of values covered is maximized, or alternately, the number of uncovered values (or outliers) is minimized. Let minOut(i, k) denote this minimum number of outliers when k intervals are used to cover values in $x_1, x_2, \ldots, x_i$. The following dynamic programming relationship holds for minOut(i, k). (In the third equation below, $l \geq 0$ is the smallest index for which $x_i - x_{l+1} \leq 2e_i$.)

$$\text{minOut}(i,k) = \begin{cases} 0 & \text{if } i = 0 \\ i & \text{if } k = 0 \\ \min\{\text{minOut}(i-1,k)+1, \\ \qquad \text{minOut}(l,k-1)\} & \text{otherwise.} \end{cases}$$

The second condition essentially states that with 0 intervals, the number of outliers in $x_1, \ldots, x_i$ is at least i. The final condition corresponds to the two cases for x_i: (1) x_i does not belong to any of the k intervals (and is thus an outlier), and (2) x_i belongs to one of the k intervals.

We are now in a position to prove the following theorem that lays the groundwork for us to compute a good estimate for $L(N)$ in terms of minOut defined above.

THEOREM 3.1. *For a leaf N that still remains to be expanded, a lower bound on the cost of a fully expanded subtree with k splits and rooted at N is at least $2k + 1 + k\log(|\mathcal{X}_i|) + (k + 1 + minOut(r, k+1))\log(|dom(X_i)|)$.* ∎

Proof: A subtree with k splits has k internal nodes and $k+1$ leaves. Thus, the cost of specifying the type for each node is 1 bit and the cost of specifying the splitting attribute for each internal node is $\log(|\mathcal{X}_i|)$, resulting in a total cost of $2k+1+k\log(|\mathcal{X}_i|)$. Further, specifying the labels for the $k+1$ leaves requires at least $(k+1)\log(|dom(X_i)|)$ bits. Finally, the number of outliers in any subtree rooted at N and containing $k+1$ leaves is at least minOut$(r, k+1)$, the minimum number of outliers when the r values in N can be covered by $k+1$ arbitrary intervals of width $2e_i$. ∎

In Figure 3, we present the procedure for computing $L(N)$ for each "still to be expanded" leaf N in the partial tree R. Procedure **LowerBound** repeatedly applies Theorem 3.1 to compute lower bounds on the cost of subtrees containing 0 to b splits (for a fixed, user-specified constant b), and then returns the minimum from among them. In Steps 1–11, the procedure computes minOut values for 1 to $b+1$ intervals using the dynamic-programming relationship for minOut presented earlier. Then, **LowerBound** sets $L(N)$ to be the minimum cost from among subtrees containing at most b splits (Steps 13–14) and greater than b splits (Step 15). Note that $2b+3+(b+1)\log(|\mathcal{X}_i|)+(b+2)\log(|dom(X_i)|)$ is a lower bound on the cost of any subtree containing more than b splits.

It is straightforward to observe that the time complexity of procedure **LowerBound** is $O(rb)$. This is due to the two for loops in Steps 6 and 9 of the procedure. Further, the procedure scales for large values of r since it makes a single pass over all the values in node N. The procedure also has very low memory requirements since for computing minOut for each i (Step 10), it only needs to store in memory minOut values for $i-1$ and l.

4. EXPERIMENTAL STUDY

In this section, we present some of our results from an extensive empirical study whose objective was to compare the quality of compression due to $\mathcal{SPARTAN}$'s model-based approach with existing syntactic (gzip) and semantic (fascicles) compression techniques. (The complete set of results can be found in [2].) We conducted a wide range of experiments with three very diverse real-life data sets in which we measured both compression ratios as well as running times for $\mathcal{SPARTAN}$. The major findings of our study can be summarized as follows.

- **Better Compression Ratios.** On all data sets, $\mathcal{SPARTAN}$ produces smaller compressed tables compared to gzip and fascicles. The compression due to $\mathcal{SPARTAN}$ is more effective for tables containing mostly numeric attributes, at times outperforming gzip and fascicles by a factor of 3 (for error tolerances of 5-10%). Even for error tolerances as low as 1%, the compression due to $\mathcal{SPARTAN}$, on an average, is 20-30% better than existing schemes.

- **Small Sample Sizes are Effective.** For the data sets, even with samples as small as 50KB (0.06% of one data set), $\mathcal{SPARTAN}$ is able to compute a good set of CaRT models that result in excellent compression ratios. Thus, using samples to build the Bayesian network and CaRT models can speed up $\mathcal{SPARTAN}$ significantly.

- **Best Algorithms for $\mathcal{SPARTAN}$ Components.** Our more sophisticated CaRT-selection algorithm based on WMIS compresses the data more effectively that the simpler greedy algorithm. Further, since $\mathcal{SPARTAN}$ spends most of its time building CaRTs (between 50% and 75% depending on the data set), the integrated pruning and building of CaRTs results in significant speedups to $\mathcal{SPARTAN}$'s execution times.

Thus, our experimental results validate the thesis of this paper that $\mathcal{SPARTAN}$ is a viable and effective system for compressing massive tables. All experiments reported in this section were performed on a multi-processor (4 700MHz Pentium processors) Linux server with 1 GB of main memory.

4.1 Experimental Testbed and Methodology

Compression Algorithms. We consider three compression algorithms in our study.

- *Gzip.* gzip is the widely used lossless compression tool based on the Lempel-Ziv dictionary-based compression technique [20; 21]. We compress the table *row-wise* using gzip after doing a lexicographic sort of the table. We found this to significantly outperform the cases in which gzip was applied to a row-wise expansion of the table (without the lexicographic sort).

- *Fascicles.* In [11], Jagadish, Madar and Ng, describe two algorithms, *Single-k* and *Multi-k*, for compressing a table using fascicles. They recommend the *Multi-k* algorithm for small values of k (the number of compact attributes in the fascicle), but the *Single-k* algorithm otherwise. In our implementation, we use the *Single-k* algorithm as described in [11]. The two main input parameters to the algorithm are the number of compact attributes, k, and the maximum number of fascicles to be built for compression, P. In our experiments, for each individual data set, we used values of k and P that resulted in the best compression due to the fascicle algorithm. We found the *Single-k* algorithm to be relatively insensitive to P (similar to the finding reported in [11]) and chose P to be 500 for all three data sets. However, the sizes of the compressed tables output by *Single-k* did vary for different values of k and so for the Corel, Forest-cover and Census data sets (described below), we set k to 6, 36 and 9, respectively. Note that these large values of k justify our use of the *Single-k* algorithm. We also set the minimum size m of a fascicle to 0.01% of the data set size. For each numeric attribute, we set the compactness tolerance to two times the input error tolerance for that attribute. However, since for categorical attributes, the fascicle error semantics differs from ours, we used a compactness tolerance of 0 for every categorical attribute.

- $\mathcal{SPARTAN}$. The results shown here were obtained using our WMIS-based CaRT-selection algorithm and using the *Single-k* fascicle algorithm for the ROWAGGREGATOR component [2]. Our implementation also employs our integrated building and pruning algorithms in the CARTBUILDER component, and using a simple lower bound of $1+\min\{\log(|\mathcal{X}_i|), \log(|dom(X_i)|)\}$ for every "yet to be expanded" leaf node. In order to be fair in our comparison with fascicles, we set the error tolerance for categorical attributes to always be 0.

Real-life Data Sets. We experimented with a number of real-life data sets with very different characteristics. Due to space constraints, however, we only present results for the *Corel*[3] data set in this paper, which we found to be representative of our overall results (see [2] for the full set of experiments). This data set contains image features extracted from a Corel image collection. We used a 10.5 MB subset of the data set which contains the color histogram features of 68,040 photo images. This data set consists of 32 numerical attributes and contains 68,040 tuples.

Default Parameter Settings. The critical input parameter to the compression algorithms is the error tolerance for numeric attributes (note that we use an error tolerance of 0 for all categorical attributes). The error tolerance for a numeric attribute X_i is specified as a percentage of the width of the range of X_i-values in the table. Another important parameter to $\mathcal{SPARTAN}$ is the size of the sample that is used to select the CaRT models in the final compressed table. For these two parameters, we use default values of 1% (for error tolerance) and 50KB (for sample size), respectively, in all our experiments.

4.2 Experimental Results

Effect of Error Threshold on Compression Ratio. Figure 4(a) depicts the compression ratios for gzip, fascicles and $\mathcal{SPARTAN}$ for the Corel data set. From the figures, it is clear that $\mathcal{SPARTAN}$ outperforms both gzip and fascicles, on an average, by 20-30% on all data sets, even for a low error threshold value of 1%. The compression due to $\mathcal{SPARTAN}$ is especially striking for the Corel data set that contains only numeric attributes. For high error tolerances (e.g., 5-10%), $\mathcal{SPARTAN}$ produces a compressed Corel table that is almost a factor of 3 smaller than the compressed tables generated by gzip and fascicles, and a factor of 10 smaller than the uncompressed Corel table.

The reason gzip does not compress the data sets as well is that unlike fascicles and $\mathcal{SPARTAN}$ it treats the table simply as a sequence of bytes and is completely oblivious of the error bounds for attributes. In contrast, both fascicles and $\mathcal{SPARTAN}$ exploit data dependencies between attributes and also the semantics of error tolerances for attributes. Further, compared to fascicles which simply cluster tuples with approximately equal attribute values, CaRTs are much more sophisticated at capturing dependencies between attribute columns. This is especially true when tables contain numeric attributes since CaRTs employ semantically rich split conditions for numeric attributes like $X_i > v$. Another crucial difference between fascicle- and CaRT-based compression is that, when fascicles are used for compression, each tuple and as a consequence, every attribute value of a tuple is assigned to a single fascicle. However, in $\mathcal{SPARTAN}$, a predictor attribute and thus a predictor attribute value (belonging to a specific tuple) can be used in a number of different CaRTs to infer values for multiple different predicted attributes. Thus, CaRTs offer a more powerful and flexible model for capturing attribute correlations than fascicles. As a result, a set of CaRT predictors are able to summarize complex data dependencies between attributes much more succinctly than a set of fascicles. For an error constraint of 1%, the final Corel $\mathcal{SPARTAN}$-compressed table contains 20 CaRTs that along with outliers, consume only 1.98 MB or 18.8% of the uncompressed table size. The compression ratios for $\mathcal{SPARTAN}$ are even more impressive for larger values of error tolerance (e.g., 10%) since the storage overhead of CaRTs + outliers is even smaller at these higher error values. For example, at 10% error, in the compressed Corel data set, CaRTs consume only 0.6 MB or 5.73% of the original table size.

[3]See http://kdd.ics.uci.edu/databases/-CorelFeatures/CorelFeatures.html.

Effect of Error Threshold and Sample Size on Running Time. In Figures 4(b) and 4(c), we plot the running times for $\mathcal{SPARTAN}$ for a range of error threshold values and sample sizes. Two trends in the figures that are straightforward to observe are that $\mathcal{SPARTAN}$'s running time decreases for increasing error bounds, and increases for larger sample sizes. The reason for the decrease in execution time when the error tolerance is increased is that for larger error thresholds, CaRTs contain fewer nodes and so CaRT construction times are smaller. For instance, CaRT construction times (which constitute approximately 50-75% of $\mathcal{SPARTAN}$'s total execution time) reduce by approximately 25% as the error bound increases from 0.5% to 10%. Note the low running times for $\mathcal{SPARTAN}$ on the Corel data set.

In Figure 4(c), we plot $\mathcal{SPARTAN}$'s running time against the random sample size instead of the data set size because $\mathcal{SPARTAN}$'s DEPENDENCYFINDER and CARTBUILDER components which account for most of $\mathcal{SPARTAN}$'s running time (on an average, 20% and 75%, respectively) use the sample for model construction. $\mathcal{SPARTAN}$ makes very few passes over the entire data set (e.g., for sampling, for identifying outliers in the data set for each selected CaRT and for compressing T' using fascicles), the overhead of which is negligible compared to the overhead of CaRT model selection. Observe that $\mathcal{SPARTAN}$'s performance scales almost linearly with respect to the sample size.

Finally, in experiments with building regression trees on the data sets, we found that integrating the pruning and building phases can result in significant reductions in $\mathcal{SPARTAN}$'s running times. This is because, integrating the pruning and building phases causes fewer regression tree nodes to be expanded (since nodes that are going to be pruned later are not expanded), and thus improves CaRT building times by as much as 25%.

5. RELATED WORK

Popular compression programs (e.g., gzip, compress) employ the Lempel-Ziv algorithm [20; 21] which treats the input data as a byte string and performs lossless compression on the input. Thus, these compression routines when applied to massive tables, do not exploit data semantics or permit errors in the compressed data.

Other lossless compression schemes primarily for numeric attributes and that do not exploit correlations between attributes have recently been proposed in the database literature [8; 13]. Goldstein, Ramakrishnan and Shaft [8] propose a page level algorithm for compressing tables. For each numeric attribute, its minimum value occuring in tuples in the page is stored separately once for the entire page. Further, instead of storing the original value for the attribute in a tuple, the difference between the original value and the minimum is stored in the tuple. Thus, since storing the difference consumes fewer bits, the storage space overhead of the table is reduced. Tuple Differential Coding (TDC) [13] is a compression method that also achieves space savings by storing differences instead of actual values for attributes. However, for each attribute value in a tuple,

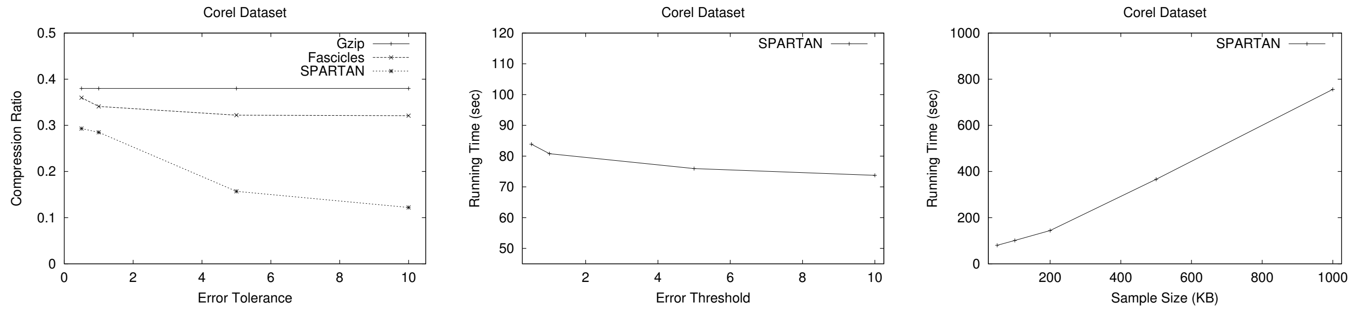


Figure 4: Effect of Error Threshold and Sample Size on Compression Ratio/Running Time.

the stored difference is relative to the attribute value in the preceding tuple.

Buchsbaum et al. [4] devise a lossless compression scheme that essentially partitions the set of attributes of table T into groups of correlated attributes that compress well (by examining a small amount of training material) and then simply uses gzip to compress the projection of T on each group. A different approach for lossless compression, proposed by Davies and Moore [5], first constructs a Bayesian network on the attributes of the table and then rearranges the table's attributes in an order that is consistent with a topological sort of the Bayesian network graph. The key intuition is that reordering the data (using the Bayesian network) results in correlated attributes being stored in close proximity; consequently, tools like gzip yield better compression ratios for the reordered table.

Another instance of a lossless compression algorithm for categorical attributes is the one proposed by Goh et al. [7]. The algorithm uses data mining techniques (e.g., classification trees, frequent itemsets) to find sets of categorical attribute values that occur frequently in the table. The frequent sets are stored separately (as rules) and occurrences of each frequent set in the table are replaced by the rule identifier for the set. The notion of a fascicle [11] generalizes the approach of Goh et al. to both numeric as well as categorical attributes and performs lossy data compression by allowing bounded errors in the compressed table.

6. CONCLUSIONS

In this paper, we have described the design and algorithms underlying $\mathcal{SPARTAN}$, a novel system that exploits attribute semantics and data-mining models to effectively compress massive data tables. $\mathcal{SPARTAN}$ takes advantage of predictive correlations between the table attributes and the user- or application-specified error-tolerance constraints to construct concise and accurate CaRT models for entire columns of the table. To restrict the huge search space of possible CaRTs, $\mathcal{SPARTAN}$ explicitly identifies strong dependencies in the data by constructing a Bayesian network model on the given attributes, which is then used to guide the selection of promising CaRT models through novel optimization algorithms. $\mathcal{SPARTAN}$'s CaRT-building component also relies on novel integrated pruning strategies that take advantage of the prescribed (per-attribute) error constraints to minimize the computational effort involved. Our experimentation with several real-life data sets has offered convincing evidence of the effectiveness of $\mathcal{SPARTAN}$'s model-based approach – $\mathcal{SPARTAN}$ has been able to consistently yield substantially better compression ratios than existing semantic or syntactic compression tools while utilizing only small samples of the data for model inference.

7. REFERENCES

[1] "NetFlow Services and Applications". Cisco Systems, 1999.

[2] S. Babu, M. Garofalakis, and R. Rastogi. "SPARTAN: A Model-Based Semantic Compression System for Massive Data Tables". In *Proc. of the 2001 ACM SIGMOD Intl. Conf. on Management of Data*, May 2001.

[3] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *"Classification and Regression Trees"*. Chapman & Hall, 1984.

[4] A.L. Buchsbaum, D.F. Caldwell, K. Church, G.S. Fowler, and S. Muthukrishnan. "Engineering the Compression of Massive Tables: An Experimental Approach". In *Proc. of the 11th Annual ACM-SIAM Symp. on Discrete Algorithms*, January 2000.

[5] S. Davies and A. Moore. "Bayesian Networks for Lossless Dataset Compression". In *Proc. of the 5th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, 1999.

[6] M.R. Garey and D.S. Johnson. *"Computers and Intractability: A Guide to the Theory of NP-Completeness"*. W.H. Freeman, 1979.

[7] C. Goh, K. Aisaka, M. Tsukamoto, K. Harumoto, and S. Nishio. "Data Compression with Data Mining Methods". In *Proc. of the 5th Intl. Conf. on Foundations of Data Organization (FODO)*, November 1998.

[8] J. Goldstein, R. Ramakrishnan, and U. Shaft. "Compressing Relations and Indexes". In *Proc. of the 14th Intl. Conf. on Data Engineering*, February 1998.

[9] M.M. Halldórsson. "Approximations of Weighted Independent Set and Hereditary Subset Problems". *Journal of Graph Algorithms and Applications*, 4(1):1–16, 2000.

[10] D.S. Hochbaum, editor. *"Approximation Algorithms for NP-Hard Problems"*. PWS Publishing Company, 1997.

[11] H.V. Jagadish, J. Madar, and R. Ng. "Semantic Compression and Pattern Extraction with Fascicles". In *Proc. of the 25th Intl. Conf. on Very Large Data Bases*, September 1999.

[12] Y. Morimoto, H. Ishii, and S. Morishita. "Efficient Construction of Regression Trees with Range and Region Splitting". In *Proc. of the 23rd Intl. Conf. on Very Large Data Bases*, August 1997.

[13] W.K. Ng and C.V. Ravishankar. "Block-Oriented Compression Techniques for Large Statistical Databases". *IEEE Trans. on Knowledge and Data Engineering*, 9(2):314–328, March 1997.

[14] Judea Pearl. *"Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference"*. Morgan Kaufmann Publishers, 1988.

[15] Judea Pearl. *"Causality – Models, Reasoning, and Inference"*. Cambridge University Press, 2000.

[16] J.R. Quinlan and R.L. Rivest. "Inferring Decision Trees Using the Minimum Description Length Principle". *Information and Computation*, 80:227–248, 1989.

[17] R. Rastogi and K. Shim. "PUBLIC: A Decision Tree Classifier that Integrates Building and Pruning". In *Proc. of the 24th Intl. Conf. on Very Large Data Bases*, August 1998.

[18] J. Shafer, R. Agrawal, and M. Mehta. "SPRINT: A Scalable Parallel Classifier for Data Mining". In *Proc. of the 22nd Intl. Conf. on Very Large Data Bases*, September 1996.

[19] W. Stallings. *"SNMP, SNMPv2, SNMPv3, and RMON 1 and 2"*. Addison-Wesley Longman, Inc., 1999. (Third Edition).

[20] J. Ziv and A. Lempel. "A Universal Algorithm for Sequential Data Compression". *IEEE Trans. on Information Theory*, 23(3), 1977.

[21] J. Ziv and A. Lempel. "Compression of Individual Sequences via Variable-rate Coding". *IEEE Trans. on Information Theory*, 24(5), 1978.