SPARTAN: A Model-Based Semantic Compression System for Massive Data Tables

Shivnath Babu*
Stanford University
shivnath@cs.stanford.edu

Minos Garofalakis
Bell Laboratories
minos@bell-labs.com

Rajeev Rastogi
Bell Laboratories
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 SPARTAN, a system that takes advantage of attribute semantics and data-mining models to perform lossy compression of massive data tables. SPARTAN is based on the novel idea of exploiting predictive data correlations and prescribed error tolerances for individual attributes to construct concise and accurate *Classification and Regression Tree (CaRT)* models for entire columns of a table. More precisely, SPARTAN selects a certain subset of 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. To restrict the huge search space and construction cost of possible CaRT predictors, SPARTAN employs sophisticated learning techniques and novel combinatorial optimization algorithms. Our experimentation with several real-life data sets offers convincing evidence of the effectiveness of SPARTAN's model-based approach – SPARTAN is able to consistently yield substantially better compression ratios than existing semantic or syntactic compression tools (e.g., gzip) while utilizing only small data samples 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 [3]. 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 [17]. 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.

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 problems, effectively compressing massive tables presents a host of novel challenges due to several distinct characteristics of table data sets and their analysis.

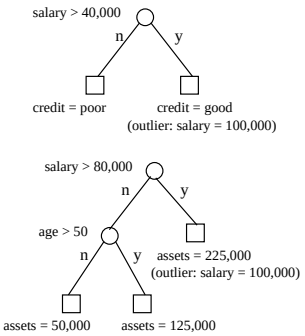
• **Approximate (Lossy) Compression.** Due to the exploratory nature of many data-analysis applications, there are several scenarios in which an exact answer may not be required, and analysts may in fact prefer a fast, approximate answer, as long as the system can guarantee an *upper bound on the error of the approximation.* 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

*Work done while visiting Bell Laboratories.

ACM SIGMOD 2001 May 21-24, Santa Barbara, California, USA

age	*salary*	*assets*	*credit*
20	30,000	25,000	poor
25	76,000	75,000	good
30	90,000	200,000	good
40	100,000	175,000	poor
50	110,000	250,000	good
60	50,000	150,000	good
70	35,000	125,000	poor
75	15,000	100,000	poor

(a) Tuples in Table



(b) CaRT Models

Figure 1: Model-Based Semantic Compression.

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.

Our Contributions. 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. $\mathcal{SPARTAN}$ 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 [2] 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 [2].

[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.

EXAMPLE 1.1. *Consider the table with 4 attributes and 8 tuples shown in Figure 1(a). Also, suppose that the acceptable errors due to compression for the numeric attributes age, salary, and assets are 2, 5,000, and 25,000, respectively. Figure 1(b) depicts a classification tree for predicting the credit attribute (with salary as the predictor attribute) and a regression tree for predicting the assets attribute (with salary and age as the predictor attributes). Observe that in the regression tree, the predicted value of assets (label value at each leaf) is almost always within 25,000, the specified error tolerance, of the actual tuple value. For instance, the predicted value of assets for the tuple with salary = 90,000 is 225,000 while the original value is 200,000. The only tuple for which the predicted value violates this error bound is the tuple with salary = 100,000, which is an marked as an outlier value in both trees. Thus, 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 (age and salary), we can ensure that the error due to compression does not exceed the user-specified bounds. Further, storing the CaRT models (plus outliers) for credit and assets instead of the attribute values themselves results in a reduction from 8 to 4 values for credit (2 labels for leaves + 1 split value at internal node + 1 outlier) and a reduction from 8 to 6 values for assets (3 labels for leaves + 2 split values at internal nodes + 1 outlier).* ∎

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 bounds. 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 [2, 10, 13]. 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]. 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 predictive structure from data. To control the computational 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 [9], 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 Tolerances.** 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 for the predicted attribute. $\mathcal{SPARTAN}$ then uses the CaRTs built to compress the full data set *in one pass*.

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.

2. OVERVIEW OF APPROACH

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 bounds are also employed in the fascicles framework [12].) 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 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

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 bounds 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 $T_c = < T', \{\mathcal{M}_1, \ldots, \mathcal{M}_p\} >$ where (1) T' is a small (possibly empty) projection of the data values in T that are retained *accurately* in T_c; and, (2) $\{\mathcal{M}_1, \ldots, \mathcal{M}_p\}$ 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 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 multi-attribute table T and a vector of (per-attribute) error tolerances $\bar{e}$, find a set of models $\{\mathcal{M}_1, \ldots, \mathcal{M}_m\}$ and a compression scheme for T based on these models $T_c =< T', \{\mathcal{M}_1, \ldots, \mathcal{M}_p\} >$ 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 [12]. (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[2].)

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 2, 5,000 and 25,000 for the three numeric attributes age, salary and assets, respectively, result in the following two fascicles:*

F_1

30	*90,000*	**200,000**	good
50	*110,000*	**250,000**	good

F_2

70	*35,000*	**125,000**	poor
75	*15,000*	**100,000**	poor

The tuples in the two fascicles F_1 and F_2 are similar (with respect to the permissible errors) on the asset and credit attributes (shown in bold). The reason for this is that two attribute values are considered to be similar if the difference between them is at most twice the error bound for the attribute. Thus, substituting for each attribute value, the mean of the maximum and minimum value of the attribute ensures that the introduced error is acceptable. Consequently, in order to compress the table using fascicles, the single (sub)tuple (225,000, good) replaces the two corresponding (sub)tuples in the first fascicle and (112,500, poor) is used instead of the two subtuples in the second fascicle. Thus, in the final compressed table, the maximum error for assets is 25,000, and the number of values stored for the assets and credit attributes is reduced from 8 to 6. ∎

[2] Due to space constraints, we omit a detailed discussion of related work; it can be found in the full version of this paper [1].

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, different classes of data-mining models (like Classification and Regression Trees (CaRTs)) can accurately capture and model such correlations and, thereby, attain much better semantic compression in such scenarios.

Revisiting Example 1.1, the two CaRTs in Figure 1(b) can be used to predict values for the *assets* and *credit* attributes, thus completely eliminating the need to explicitly store values for these attributes. Note that CaRTs result in better compression ratios than fascicles for our example table – the storage for the *credit* attribute reduces from 8 to 4 with CaRTs compared to 6 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 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 [12]. 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 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 set 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 $\mathcal{X}_i \rightarrow X_i$ 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 $\mathcal{X}_i \rightarrow X_i$ is denoted by $\texttt{PredCost}(\mathcal{X}_i \rightarrow X_i)$; 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. The issues involved in combining our CaRT-based compression methodology with row-wise clustering techniques are addressed in more detail later in the paper.

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; we defer a more detailed description of each component and the relevant algorithms to Section 3.

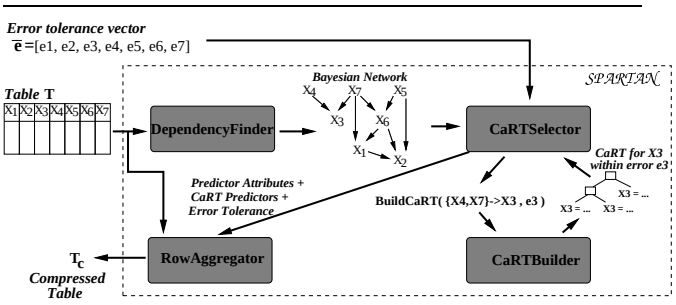


Figure 2: $\mathcal{SPARTAN}$ System Architecure.

• DEPENDENCYFINDER. 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. (The exact dependence semantics of G are defined shortly.) Thus, intuitively, a set of nodes in the "neighborhood" of 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.

• CARTSELECTOR. 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 correponding 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 [9]. 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 [11] 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 $\mathcal{X}_i \rightarrow X_i$, 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 [2, 13]. As we demonstrate, however, $\mathcal{SPARTAN}$'s CaRT construction algorithms can take advantage of the compression semantics and exploit the user-defined error-tolerances 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 [12] 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.

3. SPARTAN SYSTEM COMPONENTS

3.1 The DEPENDENCYFINDER Component

Motivation. As explained in Section 2.2, the essence of SPARTAN's CaRT-based semantic compression problem lies in discovering a collection of "strong" predictive correlations among the attributes of an arbitrary table. The search space for this problem is obviously exponential: given any attribute X_i, *any subset* of $\mathcal{X} - \{X_i\}$ could potentially be used to construct a predictor for X_i! Furthermore, verifying the quality of a predictor for the purposes of semantic compression is typically a computation-intensive task, since it involves actually building the corresponding classification or regression tree on the given subset of attributes [2, 10, 13]. Since building an exponentially large number of CaRTs is clearly impractical, we need a methodology for producing a concise *interaction model* that identifies the strongest predictive correlations among the input attributes. This model can then be used to restrict the search to interesting regions of the prediction space, limiting CaRT construction to truly promising predictors. Building such an interaction model is the main purpose of SPARTAN's DEPENDENCYFINDER component.

The specific class of attribute interaction models used in the current SPARTAN implementation is that of *Bayesian networks* [14]. Briefly, a Bayesian network is a combination of a probability distribution and a structural model in the form of a DAG over the attributes in which edges represent direct probabilistic dependence. 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 [16]; such interpretations are typically based on the following dependence semantics of the Bayesian network structure.

• *Parental Markov Condition [14]:* Given a Bayesian network G over a set of attributes $\mathcal{X}$, any node $X_i \in \mathcal{X}$ is independent of all its non-descendant nodes given its parent nodes in G (denoted by $\pi(X_i)$).

• *Markov Blanket Condition [14]:* Given a Bayesian network G over a set of attributes $\mathcal{X}$, we define the *Markov blanket* of $X_i \in \mathcal{X}$ (denoted by $\beta(X_i)$) as the union of X_i's parents, X_i's children, and the parents of X_i's children in G. Any node $X_i \in \mathcal{X}$ is independent of all other nodes given its Markov blanket in G.

Based on the above conditions, a Bayesian network over the attributes of the input table can provide definite guidance on the search for promising CaRT predictors for semantic compression. More specifically, it is clear that predictors of the form $\pi(X_i) \rightarrow X_i$ or $\beta(X_i) \rightarrow X_i$ should be considered as prime candidates for CaRT-based semantic compression.

Construction Algorithm. Learning the structure of Bayesian networks from data is a difficult problem that has seen growing research interest in recent years [4, 6, 8]. There are two general approaches to discovering Bayesian structure: (1) *Constraint-based methods* try to discover conditional independence properties between data attributes using appropriate statistical measures (e.g., χ^2 or mutual information) and then build a network that exhibits the observed correlations and independencies [4, 16]. (2) *Scoring-based (or, Bayesian) methods* are based on defining a statistically-motivated *score function* (e.g., Bayesian or MDL-based) that describes the fitness of a probabilistic network structure to the observed data; the goal then is to find a structure that maximizes the score [6, 7, 8]. (In general, this is a hard optimization problem that is typically $\mathcal{NP}$-hard [5].) Both methods have their pros and cons. Given the intractability of scoring-based network generation, several heuristic search methods with reasonable time complexities have been proposed. Many of these scoring-based methods, however, assume an *ordering* for the input attributes and can give drastically different networks for different attribute orders. Further, due to their heuristic nature, such heuristic methods may not find the best structure for the data. On the other hand, constraint-based methods have been shown to be asymptotically correct under certain assumptions about the data [4], but, typically, introduce edges in the network based on Conditional Independence (CI) tests that become increasingly expensive and unreliable as the size of the conditioning set increases [7]. Also, several constraint-based methods have very high computational complexity, requiring, in the worst case, an exponential number of CI tests.

SPARTAN's DEPENDENCYFINDER implements a constraint-based Bayesian network builder based on the algorithm of Cheng et al. [4]. Unlike earlier constraint-based methods, the algorithm of Cheng et al. explicitly tries to avoid complex CI tests with large conditioning sets and, by using CI tests based on mutual information divergence, eliminates the need for an exponential number of CI tests [4]. In fact, given an n-attribute data set, our Bayesian network builder requires at most $O(n^4)$ CI tests, which, in our implementation, translates to at most $O(n^4)$ passes over the input tuples. Recall that SPARTAN's DEPENDENCYFINDER uses only a small random sample of the input table to discover the attribute interactions; the size of this sample can be adjusted according to the amount of main memory available, so that no I/O is incurred (other than that required to produce the sample). Also, note that the DEPENDENCYFINDER is, in a sense, out of the "critical path" of the data compression process, since such attribute interactions are an intrinsic characteristic of the data semantics that only needs to be discovered *once* for each input table. Our DEPENDENCYFINDER implementation adds several enhancements to the basic Cheng et al. algorithm, such as the use of Bayesian-scoring methods for appropriately orienting the edges in the final network [1].

3.2 The CARTSELECTOR Component

The CARTSELECTOR component is the heart of SPARTAN's model-based semantic compression engine. Given the input data table and error tolerances, as well as the Bayesian network capturing the attribute interactions, the goal of the CARTSELECTOR is to select (1) a subset of attributes to be predicted and (2) the corresponding CaRT-based predictors, such that the overall storage cost is minimized within the specified error bounds. Recall from Section 2.2 that the total storage cost $|T_c|$ is the sum of the materialization costs (of predictor attributes) and prediction costs (of the CaRTs for predicted attributes). In essence, the CARTSELECTOR implements the core algorithmic strategies for solving SPARTAN's CaRT-based semantic compression problem (Section 2.2). Deciding on a storage-optimal set of predicted attributes and corresponding predictors poses a hard combinatorial optimization problem; as the following theorem shows, the problem is $\mathcal{NP}$-hard even in the simple case where the set of predictor attributes to be used for each attribute is fixed.

THEOREM 3.1. *Consider a given set of n predictors $\{\mathcal{X}_i \rightarrow X_i : \text{for all } X_i \in \mathcal{X}, \text{ where } \mathcal{X}_i \subseteq \mathcal{X}\}$. Choosing a storage-optimal subset of attributes $\mathcal{X}_{pred} \subseteq \mathcal{X}$ to be predicted using attributes in $\mathcal{X} - \mathcal{X}_{pred}$ is $\mathcal{NP}$-hard.* ∎

The simple instance of SPARTAN's CaRT-based semantic compression problem described in the above theorem can be shown to

be equivalent to the *Weighted Maximum Independent Set (WMIS)* problem, which is known to be $\mathcal{NP}$-hard. The WMIS problem can be stated as follows: "Given a node-weighted, undirected graph $G = (V, E)$, find a subset of nodes $V' \subseteq V$ such that no two vertices in V' are joined by an edge in E and the total weight of nodes in V' is maximized." Abstractly, the partitioning of the nodes into V' and $V - V'$ corresponds exactly to the partitioning of attributes into "predicted" and "materialized" with the edges of G capturing the "predicted by" relation. Further, the constraint that no two vertices in V' are adjacent in G ensures that all the (predictor) attributes for a predicted attribute (in V') are materialized, which is a requirement of $\mathcal{SPARTAN}$'s compression problem. Also, the weight of each node coresponds to the "storage benefit" (materialization cost - prediction cost) of predicting the corresponding attribute. Thus, maximizing the storage benefit of the predicted attributes has the same effect as minimizing the overall storage cost of the compressed table.

Even though WMIS is known to be $\mathcal{NP}$-hard and notoriously difficult to approximate for general graphs [9], several recent approximation algorithms have been proposed with guaranteed worst-case performance bounds for *bounded-degree graphs* [11]. The optimization problem faced by $\mathcal{SPARTAN}$'s CARTSELECTOR is obviously much harder than simple WMIS, since the CARTSELECTOR is essentially free to decide on the set of predictor attributes for each CaRT. Further, the CARTSELECTOR also has to invoke $\mathcal{SPARTAN}$'s CARTBUILDER component to actually build potentially useful CaRTs, and this construction is itself a computation-intensive task [2, 13].

Given the inherent difficulty of the CaRT-based semantic compression problem, $\mathcal{SPARTAN}$'s CARTSELECTOR implements two distinct heuristic search strategies that employ the Bayesian network model of T built by the DEPENDENCYFINDER to intelligently guide the search through the huge space of possible attribute prediction alternatives. The first strategy is a simple *greedy* selection algorithm that chooses CaRT predictors greedily based on their storage benefits during a single "roots-to-leaves" traversal of the Bayesian graph. The second, more complex strategy takes a less myopic approach that exploits the similarities between our CaRT-selection problem and WMIS; the key idea here is to determine the set of predicted attributes (and the corresponding CaRTs) by obtaining (approximate) solutions to a number of WMIS instances created based on the Bayesian model of T.

The Greedy CaRT Selector. Briefly, $\mathcal{SPARTAN}$'s **Greedy** CaRT-selection algorithm visits the set of attributes $\mathcal{X}$ in the topological-sort order imposed by the constructed Bayesian network model G and tries to build a CaRT predictor for each attribute based on its predecessors. Thus, for each attribute X_i visited, there are two possible scenarios.

1. If X_i has no parent nodes in G (i.e., node X_i is a root of G) then **Greedy** concludes that X_i cannot be predicted and, consequently, places X_i directly in the subset of materialized attributes $\mathcal{X}_{mat}$.

2. Otherwise (i.e., $\pi(X_i)$ is not empty in G), $\mathcal{SPARTAN}$'s CARTBUILDER component is invoked to construct a CaRT-based predictor for X_i (within the specified error tolerance e_i) using the set of attributes chosen for materialization thus far $\mathcal{X}_{mat}$. (Note that possibly irrelevant attributes in $\mathcal{X}_{mat}$ will be filtered out by the CaRT construction algorithm in CARTBUILDER.) Once the CaRT for X_i is built, the relative storage benefit of predicting X_i can be estimated; X_i is chosen for prediction if this relative benefit is at least θ (an input parameter to **Greedy**) and materialized otherwise.

Our **Greedy** algorithm provides a simple, low-complexity solution to $\mathcal{SPARTAN}$'s CaRT-based semantic compression problem. (The detailed pseudo-code for **Greedy** can be found in [1].) Given an n-attribute table and Bayesian network G, it is easy to see that **Greedy** always constructs *at most* $(n-1)$ CaRT predictors during its traversal of G. This simplicity, however, comes at a price. More specifically, **Greedy** CaRT selection suffers from two major shortcomings. First, selecting an attribute X_i to be predicted based solely on its "localized" prediction benefit (through its predecessors in G) is a very myopic strategy, since it ignores the potential benefits from using X_i itself as a (materialized) predictor attribute for its descendants in G. Such very localized decisions can obviously result in poor overall predictor selections. Second, the value of the "benefit threshold" parameter θ can adversely impact the performance of the compression engine and selecting a reasonable value for θ is not a simple task. A high θ value may mean that very few or no predictors are chosen, whereas a low θ value may cause low-benefit predictors to be chosen early in the search thus excluding some high-benefit predictors at lower layers of the Bayesian network.

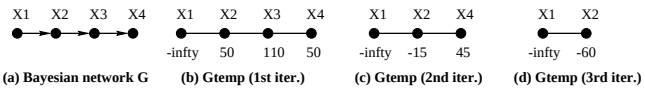


Figure 3: Example Instance for CARTSELECTOR **Algorithms.**

EXAMPLE 3.1. *Consider the Bayesian network graph defined over attributes $X_1, \ldots, X_4$ shown in Figure 3(a). Let the materialization cost of each attribute be 125. Further, let the prediction costs of CaRT predictors be as follows:*

$$\texttt{PredCost}(\{X_1\} \rightarrow X_2) = 75 \quad \texttt{PredCost}(\{X_2\} \rightarrow X_3) = 15$$
$$\texttt{PredCost}(\{X_1\} \rightarrow X_3) = 80 \quad \texttt{PredCost}(\{X_2\} \rightarrow X_4) = 80$$
$$\texttt{PredCost}(\{X_1\} \rightarrow X_4) = 125 \quad \texttt{PredCost}(\{X_3\} \rightarrow X_4) = 75$$

Suppose that $\theta = 1.5$. Since X_1 has no parents, it is initially added to $\mathcal{X}_{mat}$. In the next two iterations, since $\texttt{MaterCost}(X_2) / \texttt{PredCost}(\{X_1\} \rightarrow X_2) = 1.67 > 1.5$ *and* $\texttt{MaterCost}(X_3) / \texttt{PredCost}(\{X_1\} \rightarrow X_3) = 1.56 > 1.5$, *$X_2$ and X_3 are added to $\mathcal{X}_{pred}$. Finally, X_4 is added to $\mathcal{X}_{mat}$ since* $\texttt{MaterCost}(X_4) / \texttt{PredCost}(\{X_1\} \rightarrow X_4) = 1 < 1.5$. *Thus, the overall storage cost of materializing X_1 and X_4, and predicting X_2 and X_3 is $125 + 75 + 80 + 125 = 405$.* ∎

The MaxIndependentSet CaRT Selector. $\mathcal{SPARTAN}$'s **MaxIndependentSet** CaRT-selection algorithm, depicted in Figure 4, alleviates the drawbacks of **Greedy** mentioned above. Intuitively, the **MaxIndependentSet** algorithm starts out by assuming all attributes to be materialized, i.e., $\mathcal{X}_{mat} = \mathcal{X}$ (Step 1), and then works by iteratively solving WMIS instances that try to improve the overall storage cost by moving the nodes in the (approximate) WMIS solution to the subset of predicted attributes $\mathcal{X}_{pred}$. Consider the *first iteration* of the main while-loop (Steps 3–30). Algorithm **MaxIndependentSet** starts out by building CaRT-based predictors for each attribute X_i in $\mathcal{X}$ based on X_i's "predictive `neighborhood`" in the constructed Bayesian network G (Steps 5–7); this `neighborhood` function is an input parameter to the algorithm and can be set to either X_i's parents or its Markov blanket in G. Then, based on the "predicted-by" relations observed in the constructed CaRTs, **MaxIndependentSet** builds a node-weighted, undirected graph G_{temp} on $\mathcal{X}$ with (a) all edges (X_i, Y), where Y is used in the CaRT predictor for X_i, and (b) weights for each node

procedure MaxIndependentSet($T(\mathcal{X})$, $\bar{e}$, G , `neighborhood`())
Input: n-attribute table T; n-vector of error tolerances $\bar{e}$; Bayesian network G on the set of attributes $\mathcal{X}$; function `neighborhood`() defining the "predictive neighborhood" of a node X_i in G (e.g., $\pi(X_i)$ or $\beta(X_i)$).
Output: Set of materialized (predicted) attributes $\mathcal{X}_{mat}$ ($\mathcal{X}_{pred} = \mathcal{X} - \mathcal{X}_{mat}$) and a CaRT predictor $\texttt{PRED}(X_i) \rightarrow X_i$ for each $X_i \in \mathcal{X}_{pred}$.
begin
1. $\mathcal{X}_{mat} := \mathcal{X}$, $\mathcal{X}_{pred} := \phi$
2. $\texttt{PRED}(X_i) := \phi$ for all $X_i \in \mathcal{X}$, improve := **true**
3. **while (** improve $\neq$ **false**) **do**
4. **for each** $X_i \in \mathcal{X}_{mat}$
5. $\texttt{mater\_neighbors}(X_i) := (\mathcal{X}_{mat} \cap \texttt{neighborhood}(X_i)) \cup \{\texttt{PRED}(X) : X \in \texttt{neighborhood}(X_i), X \in \mathcal{X}_{pred}\} - \{X_i\}$
6. $\mathcal{M} := \texttt{BuildCaRT}(\texttt{mater\_neighbors}(X_i) \rightarrow X_i, e_i)$
7. let $\texttt{PRED}(X_i) \subseteq \texttt{mater\_neighbors}(X_i)$ be the set of predictor attributes used in $\mathcal{M}$
8. $\texttt{cost\_change}_i := 0$
9. **for each** $X_j \in \mathcal{X}_{pred}$ such that $X_i \in \texttt{PRED}(X_j)$
10. $\texttt{NEW\_PRED}_i(X_j) := \texttt{PRED}(X_j) - \{X_i\} \cup \texttt{PRED}(X_i)$
11. $\mathcal{M} := \texttt{BuildCaRT}(\texttt{NEW\_PRED}_i(X_j) \rightarrow X_j, e_j)$
12. set $\texttt{NEW\_PRED}_i(X_j)$ to the (sub)set of predictor attributes used in $\mathcal{M}$
13. $\texttt{cost\_change}_i := \texttt{cost\_change}_i + (\texttt{PredCost}(\texttt{PRED}(X_j) \rightarrow X_j) - \texttt{PredCost}(\texttt{NEW\_PRED}_i(X_j) \rightarrow X_j))$
14. **end**
15. **end**
16. build an undirected, node-weighted graph $G_{temp} = (\mathcal{X}_{mat}, E_{temp})$ on the current set of materialized attributes $\mathcal{X}_{mat}$, where:
17. (a) $E_{temp} := \{(X, Y) : \forall$ pair $(X, Y) \in \texttt{PRED}(X_j)$ for some X_j in $\mathcal{X}_{pred}\} \bigcup \{(X_i, Y) : \forall Y \in \texttt{PRED}(X_i)$, $X_i \in \mathcal{X}_{mat}\}$
18. (b) $\texttt{weight}(X_i) := \texttt{MaterCost}(X_i) - \texttt{PredCost}(\texttt{PRED}(X_i) \rightarrow X_i) + \texttt{cost\_change}_i$ for each $X_i \in \mathcal{X}_{mat}$
19. $S := \mathbf{FindWMIS}(G_{temp})$
20. /* select (approximate) maximum `weight` independent set in */
21. /* G_{temp} as "maximum-benefit" subset of predicted attributes */
22. **if** $(\sum_{X \in S} \texttt{weight}(X) \leq 0)$ **then** improve := **false**
23. **else** /* update $\mathcal{X}_{mat}$, $\mathcal{X}_{pred}$, and the chosen CaRT predictors */
24. **for each** $X_j \in \mathcal{X}_{pred}$
25. **if (** $\texttt{PRED}(X_j) \cap S = \{X_i\}$ **) then**
26. $\texttt{PRED}(X_j) := \texttt{NEW\_PRED}_i(X_j)$
27. **end**
28. $\mathcal{X}_{mat} := \mathcal{X}_{mat} - S$, $\mathcal{X}_{pred} := \mathcal{X}_{pred} \cup S$
29. **end**
30. **end** /* while */
end

Figure 4: The MaxIndependentSet CaRT-Selection Algorithm.

X_i set equal to the storage-cost benefit of predicting X_i (Steps 16–18). Finally, **MaxIndependentSet** finds a (near-optimal) WMIS of G_{temp} and the corresponding nodes/attributes are moved to the predicted set $\mathcal{X}_{pred}$ with the appropriate CaRT predictors (assuming the total benefit of the WMIS is positive) (Steps 19–29).

Note that in Step 5, it is possible for $\texttt{mater\_neighbors}(X_i)$ to be ϕ. This could happen, for instance, if X_i is a root of G and X_i's `neighborhood` comprises of its parents. In this case, the model $\mathcal{M}$ returned by `BuildCaRT` is empty and it does not make sense for X_i to be in the predicted set $\mathcal{X}_{pred}$. We ensure that X_i always stays in $\mathcal{X}_{mat}$ by setting $\texttt{PredCost}(\texttt{PRED}(X_i) \rightarrow X_i)$ to ∞ if $\texttt{PRED}(X_i) = \phi$, which causes X_i's weight to become $-\infty$ in Step 18.

The WMIS solution discovered after this first iteration of **MaxIndependentSet** can be further optimized, since it makes the rather restrictive assumption that an attribute can only be predicted based on its direct `neighborhood` in G. For example, consider a scenario where G contains the directed chain $\{X, Y\} \rightarrow Z \rightarrow W$, and the attribute pair $\{X, Y\}$ provides a very good predictor for Z, which itself is a good predictor for the value of W. Then, the initial WMIS solution can obviously select only one of these predictors. On the other hand, the above scenario means that (by "transitivity") it is very likely that $\{X, Y\}$ can also provide a good predictor for W (i.e., only X and Y need to be materialized).

Later iterations of **MaxIndependentSet**'s main while-loop try to further optimize the initial WMIS solution based on the above observation. This is accomplished by repeatedly moving attributes from the remaining set of materialized attributes $\mathcal{X}_{mat}$ to the predicted attributes $\mathcal{X}_{pred}$. For each materialized attribute X_i, **MaxIndependentSet** finds its "materialized neighborhood" in the Bayesian model G, that comprises for each node X in the `neighborhood` of X_i: (1) X itself, if $X \in \mathcal{X}_{mat}$ and (2) the (materialized) attributes currently used to predict X, if $X \in \mathcal{X}_{pred}$ (Step 5). A CaRT predictor for X_i based on its materialized neighbors is then constructed (Step 6). Now, since X_i may already be used in a number of predictors for attributes in $\mathcal{X}_{pred}$, we also need to account for the change in storage cost for these predictors when X_i is replaced by its materialized neighbors used to predict it; this change (denoted by $\texttt{cost\_change}_i$) is estimated in Steps 8–14. The node-weighted, undirected graph G_{temp} is then built on $\mathcal{X}_{mat}$ with the weight for each node X_i set equal to the overall storage benefit of predicting X_i, including $\texttt{cost\_change}_i$ (Steps 16-18). (Note that this benefit may very well be negative.) Finally, a (near-optimal) WMIS of G_{temp} is chosen and added to the set of predicted attributes $\mathcal{X}_{pred}$ with the appropriate updates to the set of CaRT predictors. Note that, since our algorithm considers the "transitive effects" of predicting each materialized node X_i in isolation, some additional care has to be taken to ensure that *at most one* predictor attribute from each already selected CaRT in $\mathcal{X}_{pred}$ is chosen at each iteration. This is accomplished by ensuring that all attributes belonging to a predictor set $\texttt{PRED}(X_j)$ for some $X_j \in \mathcal{X}_{pred}$ form a *clique* in the construction of G_{temp} (Step 17). Then, by its definition, the WMIS solution can contain at most one node from each such set $\texttt{PRED}(X_j)$. **MaxIndependentSet**'s while-loop continues until no further improvements on the overall storage cost are possible (Step 22).

EXAMPLE 3.2. *Consider the Bayesian network graph shown in Figure 3(a) and let prediction costs for attributes be as described earlier in Example 3.1. Further, suppose the* `neighborhood` *function for a node X_i is its parents. In the first iteration,* $\texttt{PRED}(X_1) = \phi$, $\texttt{PRED}(X_2) = X_1$, $\texttt{PRED}(X_3) = X_2$ *and* $\texttt{PRED}(X_4) = X_3$. *Further, since* $\mathcal{X}_{pred} = \phi$, $\texttt{cost\_change}_i = 0$ *for each* $X_i \in \mathcal{X}_{mat}$. *As a result, the graph G_{temp} and weights for the nodes are set as shown in Figure 3(b). Note that node X_1 is assigned a weight of $-\infty$ because* $\texttt{PRED}(X_1) = \phi$. *The optimal WMIS of G_{temp} is $\{X_3\}$ since its weight is greater than the sum of the weights of X_2 and X_4. Thus, after the first iteration* $\mathcal{X}_{pred} = \{X_3\}$.

In the second iteration, $\texttt{PRED}(X_4)$ *is set to X_2 in Steps 5–7 since* $\texttt{neighborhood}(X_4) = X_3$ *and* $X_3 \in \mathcal{X}_{pred}$ *with* $\texttt{PRED}(X_3) = X_2$. *Further,* $\texttt{PRED}(X_1) = \phi$ *and* $\texttt{PRED}(X_2) = X_1$. *Also, since* $X_2 \in \texttt{PRED}(X_3)$, *in Steps 8–14,* $\texttt{NEW\_PRED}_2(X_3) = \{X_1\}$ *and* $\texttt{cost\_change}_2 = \texttt{PredCost}(\{X_2\} \rightarrow X_3) - \texttt{PredCost}(\{X_1\} \rightarrow X_3) = -65$. *In addition, since X_1 and X_4 are not predictors for a predicted attribute,* $\texttt{cost\_change}_1 = \texttt{cost\_change}_4 = 0$. *Thus, the graph G_{temp} and weights for the nodes are set as shown in Figure 3(c). The weight for X_2 is essentially* $\texttt{MaterCost}(X_2) - \texttt{PredCost}(\{X_1\} \rightarrow X_2) + \texttt{cost\_change}_2 = 125 - 75 - 65$ *and the weight for X_4 is* $\texttt{MaterCost}(X_4) - \texttt{PredCost}(\{X_2\} \rightarrow X_4) + \texttt{cost\_change}_4 = 125 - 80 + 0$. *The optimal WMIS of G_{temp} is $\{X_4\}$ and thus $\mathcal{X}_{pred} = \{X_3, X_4\}$ after the second iteration.*

Finally, Figure 3(d) illustrates G_{temp} during the third iteration – node X_2 has a weight of -60 since $\texttt{PRED}(X_2) = \{X_1\}$ *and X_2 is*

used to predict both X_3 and X_4. Thus, while predicting (instead of materializing) X_2 results in a decrease of 50 in the cost of X_2, the cost of predicting X_3 and X_4 using X_1 (instead of X_2) increases increases by 110, thus resulting in a net increase in cost of 60. The algorithm terminates since weight of every node in G_{temp} is negative. The end result is a total storage cost of only 345, which is, in fact, the optimal solution for this instance. ∎

Complexity of Algorithm MaxIndependentSet. Analyzing the execution of algorithm **MaxIndependentSet**, we can show that, *in the worst case,* it requires at most $O(n)$ invocations of the WMIS solution heuristic (**FindWMIS**) and constructs at most $O(\rho\frac{n^2}{2})$ CaRT predictors, where n is the number of attributes in $\mathcal{X}$ and ρ is an upper bound on the number of predictor attributes used for any attribute in $\mathcal{X}_{pred}$. Further, under assumptions slightly less pessimistic than the worst case, it can be shown that our **MaxIndependentSet** algorithm only needs to solve $O(\log n)$ WMIS instances and build $O(\rho n \log n)$ CaRT predictors. The details of the analysis can be found in the full paper [1].

3.3 The CARTBUILDER Component

$\mathcal{SPARTAN}$'s CARTBUILDER component constructs a CaRT predictor $\mathcal{X}_i \rightarrow X_i$ for the attribute X_i with $\mathcal{X}_i$ as the predictor attributes. The CARTBUILDER'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 error tolerance for attribute X_i.

If the predicted attribute X_i is *categorical*, then $\mathcal{SPARTAN}$'s CARTBUILDER component builds a compact classification tree with values of X_i serving as class labels. CARTBUILDER employs classification tree construction algorithms from [15] 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 case of *numeric* predicted attributes X_i, $\mathcal{SPARTAN}$'s CARTBUILDER employs a novel, efficient algorithm for constructing compact regression trees for predicting X_i with an error that is guaranteed not to exceed e_i. The key technical idea in our algorithm is to integrate building and pruning during the top-down construction of a guaranteed-error regression tree – this is achieved through a novel technique (based on dynamic programming) for computing a lower bound on the cost of a yet-to-be-expanded subtree. Due to space constraints, the details of $\mathcal{SPARTAN}$'s regression tree construction algorithm can be found in the full paper [1].

3.4 The ROWAGGREGATOR Component

$\mathcal{SPARTAN}$'s CARTSELECTOR component computes the set of attributes $\{X_1, \ldots, X_p\}$ to predict and the CaRT models $\{\mathcal{M}_1, \ldots, \mathcal{M}_p\}$ for predicting them. These models are stored in the compressed version T_c of the table along with T', the projection of table T on predictor attributes. Obviously, by compressing T' one could reduce the storage overhead of T_c even further. However, while lossless compression algorithms can be used to compress T' without any problems, we need to be more careful when applying lossy compression algorithms to T'. This is because, with lossy compression, the value of a predictor attribute X_i in T' may be different from its original value that was initially used to build the CaRT models. As a result, it is possible for errors that exceed the specified bounds, to be introduced into the values of predicted attributes. For instance, consider the table from Example 1.1 (shown in Figure 1(a)) and the CaRTs in Figure 1(b) contained in the compressed version T_c of the table. Suppose that the error tolerance for the *salary* attribute is 5,000 and after (lossy) compression, the *salary* value of 76,000 is stored as 81,000 in T'. Consequently, since the classification tree in Figure 1(b) is used to predict the value of the *assets* attribute, the value of the attribute would be wrongly predicted as 225,000 (instead of 75,000), thus violating the error bound of 25,000.

$\mathcal{SPARTAN}$'s ROWAGGREGATOR component uses a fascicle-based algorithm [12] to further compress the table T' of predictor attributes. Since fascicle-based compression is lossy, in the following, we show how the above-mentioned scenario can be avoided when compressing numeric attributes using fascicles. For a numeric predictor attribute X_i, define value v to be a split value for X_i if $X_i > v$ is a split condition in some CaRT $\mathcal{M}_i$ in T_c. Also, in a fascicle (set of records), we say that an attribute X_i is compact if the range $[x', x'']$ of X_i-values in the fascicle, in addition to having width at most $2e_i$, also satisfies the property that for every split value v, either $x' > v$ or $x'' \leq v$. In our fascicle-based compression algorithm, for each compact attribute X_i, by using $(x' + x'')/2$ as the representative for X_i-values in the fascicle, we can ensure that the error bounds for both predictor as well as predicted attributes are respected. In fact, we can show that the values for predicted attributes are identical prior to and after T' is compressed using fascicles. This is because for each tuple t in T', the original and compressed tuple traverse the same path in every CaRT $\mathcal{M}_i$. For instance, suppose that $X_i > v$ is a split condition in some CaRT and $t[X_i]$ is different after compression. Then, if $t[X_i] > v$, it must be the case that for the fascicle containing t, for the X_i-value range $[x', x'']$, $x' > v$. Thus, the compressed value for $t[X_i]$ ($(x' + x'')/2$) must also be greater than v. In a similar fashion, we can show that when $t[X_i] \leq v$, the compressed value of $t[X_i]$ is also less than or equal to v. Thus, our more strict definition of compact attributes prevents errors in predictor attributes from rippling through the predicted attributes. Further, the fascicle computation algorithms developed in [12] can be extended in a straightforward manner to compute fascicles containing k compact attributes (according to our new definition).

4. EXPERIMENTAL STUDY

In this section, we present the results of 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. 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.** The **MaxIndependentSet** CaRT-selection algorithm compresses the data more effectively that the **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 experimental study.

• *Gzip.* gzip is the widely used lossless compression tool based on the Lempel-Ziv dictionary-based compression technique [18]. 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 [12], 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 [12]. 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 [12]) 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 our's, we used a compactness tolerance of 0 for every categorical attribute.

• $\mathcal{SPARTAN}$. We implemented all components of the $\mathcal{SPARTAN}$ system as described in section 3. For the **Greedy** CaRT-selection algorithm, we used value of 2 for the relative benefit parameter θ. In the **MaxIndependentSet** CaRT-selection algorithm, for finding the WMIS of the node-weighted graph G_{temp}, we used the *QUALEX* software package (`www.busygin.dp.ua/npc.html`). This software implements an algorithm based on a quadratic programming formulation of the maximum weighted clique problem [9]. The running time is $O(n^4)$ (where n is number of vertices in the graph). In our experiments, *QUALEX* always found the optimal solution and accounted for a negligible fraction of the overall execution time. We also implemented the integrated building and pruning algorithm in the `BuildCaRT` component, and used a simple lower bound of $1 + \min\{\log(|\mathcal{X}_i|), \log(|dom(X_i)|)\}$ for every "yet to be expanded" leaf node. Finally, in the ROWAGGREGATOR component, we employed the *Single-k* fascicle algorithm, with P set to 500 and k equal to two-thirds of the number of attributes in T'. 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 used the following real-life data sets with very different characteristics in our experiments.

• *Census.* (`www.bls.census.gov/`) This data set was taken from the Current Population Survey (CPS) data, which is a monthly survey of about 50,000 households conducted by the Bureau of the Census for the Bureau of Labor Statistics. Each month's data contains around 135,000 tuples with 361 attributes, of which we used 7 categorical attributes (e.g., education, race) and 7 numeric attributes (e.g., age, hourly pay rate). In the final data set, we used data for 5 months (June through October 2000) that contained a total of 676,000 tuples and occupied 28.6 MB of storage.

• *Corel.* (`kdd.ics.uci.edu/databases/CorelFeatures/`) 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.

• *Forest-cover.* (`kdd.ics.uci.edu/databases/covertype/`) This data set contains the forest cover type for 30×30 meter cells obtained from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. The 75.2 MB data set contains 581,000 tuples, and 10 numeric and 44 categorical attributes that describe the elevation, slope, soil type, etc. of the cells.

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. Note that 50KB corresponds to 0.065%, 0.475% and 0.174% of the total size of the Forest-cover, Corel and Census data sets, respectively. Finally, unless stated otherwise $\mathcal{SPARTAN}$ always uses **MaxIndependentSet** for CaRT-selection and the integrated pruning and building algorithm for constructing regression trees.

4.2 Experimental Results

Effect of Error Threshold on Compression Ratio. Figure 5 depicts the compression ratios for gzip, fascicles and $\mathcal{SPARTAN}$ for the three data sets. 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. Even for the Census data set, which contains an equal number of numeric and categorical attributes, $\mathcal{SPARTAN}$ compresses better than fascicles for smaller and moderate error threshold values (e.g., 0.5% to 5%); only for larger error bounds (e.g., 10%) do fascicles perform slightly better than $\mathcal{SPARTAN}$.

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

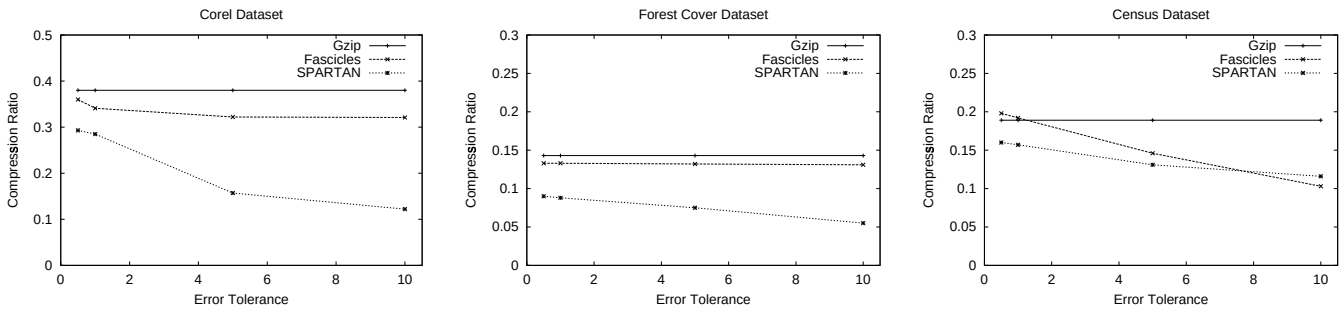


Figure 5: Effect of Error Threshold on Compression Ratio.

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. Similarly, for the Forest-cover data set, the number of predicted attributes in the compressed table is 21 (8 numeric and 13 categorical) and the CaRT storage overhead (with outliers) is a measly 4.77 MB or 6.25% of the uncompressed table.

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. Similarly, for Forest-cover, the CaRT storage overhead reduces to 2.84 MB or 3.72% of the uncompressed table. The only exception is the Census data set where the decrease in storage overhead is much steeper for fascicles than for CaRTs. We conjecture that this is because of the small attribute domains in the Census data that cause each fascicle to cover a large number of tuples at higher error threshold values.

Effect of Random Sample Size on Compression Ratio. Figure 6(a) illustrates the effect on compression ratio as the sample size is increased from 25KB to 200KB for the Forest-cover data set. Interestingly, even with a 25KB sample, which is about 0.03% of the total data set size, $\mathcal{SPARTAN}$ is able to obtain a compression ratio of approximately 0.1, which is about 25% better than the compression ratio for gzip and fascicles. Further, note that increasing the sample size beyond 50KB does not result in significant improvements in compression quality. The implication here is that it is possible to infer a good set of models even with a small random sample of the data set. This is significant since using a small sample instead of the entire data set for CaRT model construction can significantly improve $\mathcal{SPARTAN}$'s running time (see running time experiments described later).

Effect of CaRT Selection Algorithm on Compression Ratio / Running Time. In Table 1, we show the compression ratios and running times of $\mathcal{SPARTAN}$'s CaRT-selection algorithms for the three data sets. We consider three CaRT-selection algorithms – **Greedy**, **MaxIndependentSet** with the `neighborhood` for a node set to it parents and **MaxIndependentSet** with the `neighborhood` for a node set to it Markov blanket (in the Bayesian graph). From the table, it follows that the **MaxIndependentSet** algorithms always compress better than the **Greedy** algorithm. This is because the **Greedy** algorithm follows a very "local" prediction strategy for each attribute, basing the decision on whether or not to predict an attribute solely on how well it is predicted by its materialized ancestors in the Bayesian network graph. In contrast, the **MaxIndependentSet** algorithm adopts a more "global" view when making a decision on whether to predict or materialize an attribute – specifically, it not only takes into account how well an attribute is predicted by attributes in its neighborhood, but also how well it predicts other attributes in its neighborhood. Observe that, in general, the version of **MaxIndependentSet** with the Markov blanket as a node's `neighborhood` performs slightly better than **MaxIndependentSet** with parents as the `neighborhood`.

With respect to running times, in general, we found that **MaxIndependentSet** with parents performs quite well across all the data sets. This is because, it constructs few CaRTs (18 for Census, 32 for Corel and 46 for Forest-cover) and since it restricts the `neighborhood` for each attribute to only its parents, each CaRT contains few predictor attributes. While **Greedy** does build the fewest CaRTs in most cases (10 for Census, 16 for Corel and 19 for Forest-cover), all the materialized ancestors of an attribute are used as predictor attributes when building the CaRT for the attribute. As a result, since close to 50% attributes are materialized for the data sets, each CaRT is built using a large number of attributes, thus hurting **Greedy**'s performance. Finally, the performance of **MaxIndependentSet** with Markov blanket suffers since it, in some cases, constructs a large number of CaRTs (56 for Census, 17 for Corel and 138 for Forest-cover). Further, since the Markov blanket for a node contains more attributes than simply its parents, the number of predictor attributes used in each CaRT for Markov blanket is typically much larger. As a result, CaRT construction times for Markov blanket are higher and overall execution times for Markov blanket are less competitive.

Data Set	Compression Ratio/Running Time (sec)		
	Greedy	WMIS(Parent)	WMIS(Markov)
Corel	0.352 / 148.25	0.292 / 97.44	0.287 / 80.73
Forest-cover	0.131 / 932	0.106 / 670	0.1 / 1693
Census	0.18 / 205.77	0.148 / 153	0.157 / 453.35

Table 1: Effect of CaRT Selection Algorithm on Compression Ratio/Running Time.

Effect of Error Threshold and Sample Size on Running Time. In Figures 6(b) and 6(c), we plot the running times for $\mathcal{SPARTAN}$

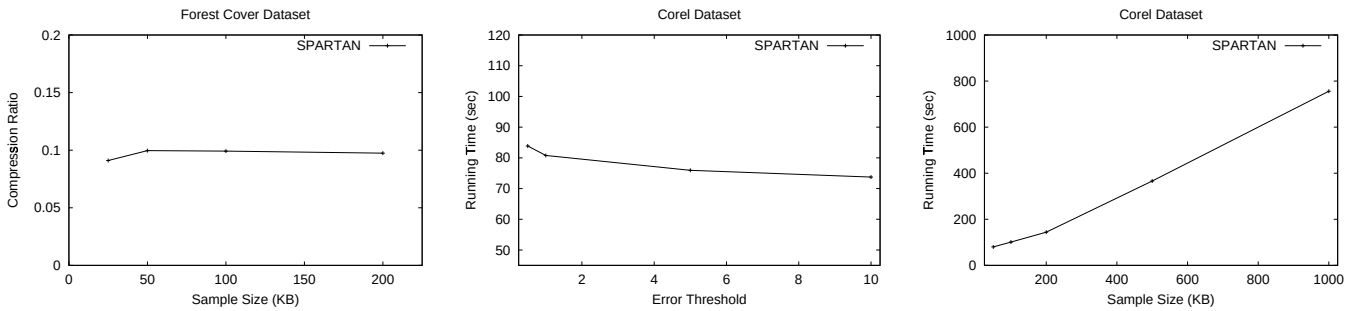


Figure 6: Effect of Error Threshold and Sample Size on Compression Ratio/Running Time.

for a range of error threshold values and sample sizes. Two trends in the figures that are straightforward to observe are that 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 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 SPARTAN on the Corel data set.

In Figure 6(c), we plot SPARTAN's running time against the random sample size instead of the data set size because SPARTAN's DEPENDENCYFINDER and CARTBUILDER components which account for most of SPARTAN's running time (on an average, 20% and 75%, respectively) use the sample for model construction. 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 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 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. CONCLUSIONS

In this paper, we have described the design and algorithms underlying SPARTAN, a novel system that exploits attribute semantics and data-mining models to effectively compress massive data tables. SPARTAN takes advantage of predictive correlations between the table attributes and the user- or application-specified error tolerances to construct concise and accurate CaRT models for entire columns of the table. To restrict the huge search space of possible CaRTs, 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. Unfortunately, as we have demonstrated in this paper, this model-selection problem is a strict generalization of an $\mathcal{NP}$-hard combinatorial problem (WMIS); thus, we have proposed a novel algorithm for SPARTAN's CaRT-selection component that exploits the discovered Bayesian structure in the data in conjunction with efficient, near-optimal WMIS heuristics. SPARTAN's CaRT-building component also employs novel integrated pruning strategies that take advantage of the prescribed error tolerances to minimize the computational effort involved. Our experimentation with several real-life data sets has offered convincing evidence of the effectiveness of SPARTAN's model-based approach.

6. REFERENCES

[1] S. Babu, M. Garofalakis, and R. Rastogi. "SPARTAN: A Model-Based Semantic Compression System for Massive Data Tables". Bell Labs Tech. Report, 2001.

[2] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *"Classification and Regression Trees"*. Chapman & Hall, 1984.

[3] 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*, 2000.

[4] J. Cheng, D. A. Bell, and W. Liu. "Learning Belief Networks from Data: An Information Theory Based Approach". In *Proc. of the 6th Intl. Conf. on Information and Knowledge Management*, 1997.

[5] D. Chickering, D. Geiger, and D. Heckerman. "Learning Bayesian Networks is NP-Hard". Technical Report MSR-TR-94-17, Microsoft Research, 1993.

[6] G. F. Cooper and E. Herskovits. "A Bayesian Method for Constructing Bayesian Belief Networks from Databases". In *Proc. of the 7th Annual Conf. on Uncertainty in AI*, 1991.

[7] G. F. Cooper and E. Herskovits. "A Bayesian Method for the Induction of Probabilisitc Networks from Data". *Machine Learning*, 9, 1992.

[8] N. Friedman, I. Nachman, and D. Peér. "Learning Bayesian Network Structure from Massive Datasets: The "Sparse Candidate" Algorithm". In *Proc. of the 15th Annual Conf. on Uncertainty in AI*, 1999.

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

[10] J. Gehrke, R. Ramakrishnan, and V. Ganti. "RainForest - A Framework for Fast Decision Tree Construction of Large Datasets". In *Proc. of the 24th Intl. Conf. on Very Large Data Bases*, 1998.

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

[12] 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*, 1999.

[13] 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*, 1997.

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

[15] 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*, 1998.

[16] P. Spirtes, C. Glymour, and R. Scheines. *"Causation, Prediction, and Search"*. Springer-Verlag NY, Inc., 1993.

[17] W. Stallings. *"SNMP, SNMPv2, SNMPv3, and RMON 1 and 2"*. Addison-Wesley Longman, Inc., 1999. (3rd Edition).

[18] J. Ziv and A. Lempel. "A Universal Algorithm for Sequential Data Compression". *IEEE Trans. on Info. Theory*, 23(3):337–343, 1977.