NetCube: A Scalable Tool for Fast Data Mining and Compression

Dimitris Margaritis
Computer Science Dept.
Carnegie Mellon University
Pittsburgh, PA 15213, U.S.A.
D.Margaritis@cs.cmu.edu

Christos Faloutsos*
Computer Science Dept.
Carnegie Mellon University
Pittsburgh, PA 15213, U.S.A.
C.Faloutsos@cs.cmu.edu

Sebastian Thrun†
Computer Science Dept.
Carnegie Mellon University
Pittsburgh, PA 15213, U.S.A.
S.Thrun@cs.cmu.edu

Abstract

We propose an novel method of computing and storing DataCubes. Our idea is to use Bayesian Networks, which can generate approximate counts for any query combination of attribute values and "don't cares." A Bayesian network represents the underlying joint probability distribution of the data that were used to generate it. By means of such a network the proposed method, NetCube, exploits correlations among attributes. Our proposed preprocessing algorithm scales linearly on the size of the database, and is thus scalable; it is also parallelizable with a straightforward parallel implementation. Moreover, we give an algorithm to estimate counts of arbitrary queries that is fast (*constant* on the database size). Experimental results show that NetCubes have fast generation and use (a few minutes preprocessing time per 100,000 records and less than a second query time), achieve excellent compression (at least 1800:1 compression ratios on real data) and have low reconstruction error (less than 5% on average). Moreover, our method naturally allows for visualization and data mining, at no extra cost.

1 The Problem

The problem of computing counts of records with desired characteristics from a database is a very common one in the area of decision support systems and data mining. A typical scenario is as follows: a customer analyst is interested in discovering groups of customers that exhibit an interesting or unusual behavior that might lead to possibly profitable insights into the company's customer behavior. In other words, a company wants to be able to *model* its customer base, and the better it is able to do that, the more insights it can obtain from the model and more profitable it has the opportunity to be. In this scenario an analyst would, through an interactive query process, request count information from the database, possibly drilling-down in interesting subsets of the database of customer information. It is obvious that it is very important that the results to these queries be returned quickly, because that will greatly facilitate the process of discovery. It is also important that they are accurate up to a reasonable degree, although it is not imperative that they are exact. The analyst wants an approximate figure of the result of the query and getting it correct down to the last digit is not necessary.

Our solution to the problem is motivated by these observations *i.e.* that we need great speed coupled with only reasonable accuracy. In the following paragraphs we show that this is true for our method through performance results. In fact, our method can fit a database of billions of records in the main memory of a single workstation. This is due to the fact that we do not use the data to answer the query but only *a model of the data*. In doing this, our paper proposes a new viewpoint on the computation of DataCubes, one that advocates the use of models of the data rather than the data

* This material is based upon work supported by the National Science Foundation under Grants No. DMS-9873442, IIS-9817496, IIS-9910606, IIS-9988876, LIS 9720374, IIS-0083148, IIS-0113089, and by the Defense Advanced Research Projects Agency under Contracts No. N66001-97-C-8517 and N66001-00-1-8936. Additional funding was provided by donations from Intel. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, DARPA, or other funding parties.

† This research is sponsored by the National Science Foundation (CAREER grant number IIS-9876136 and regular grant number IIS-9877033), and by DARPA-ATO via TACOM (contract number DAAE07-98-C-L032), which is gratefully acknowledged. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing official policies or endorsements, either expressed or implied, of the United States Government or any of the sponsoring institutions.

Proceedings of the 27th VLDB Conference,
Roma, Italy, 2001

themselves for answering DataCube queries. Having said that, the real challenge lies in how to construct a model of the data that is good enough for our purposes. For this, there are two important considerations that are relevant to the problem that we are addressing:

One, the model should be an accurate description of our data, or at the very least of these quantities derived from them that are of interest. In this problem the quantities are the counts in the database of every interesting count query that can be applied to them (*i.e.* queries with some minimum support such as 1%; other query results can be due to noise and errors in the data). Second, the model should be simple enough so that using it instead of the actual data to answer a query should not take an exorbitant amount of time or consume an enormous amount of space, more so perhaps than using the raw data themselves.

These two issues are conflicting, and the problem of balancing them is a central issue in the AI field of machine learning (which concerns itself with the development of models of data): it is always possible to describe the data (or the derived quantities we are interested in) better, or at least as well, with increasingly complex models. However, the cost of such models increases with complexity, in terms of both size to store the model parameters and time that it takes to use it for computing the relevant quantities (the query counts in our case). In this paper we chose to use Bayesian networks (**BNs**). Such models are not the only choice possible, but we picked them because they are a mature, broadly acceptable and well respected method of modeling data in the machine learning community. This acceptance and respect comes not only from their practical effectiveness, but also from their sound mathematical foundations in probability theory, as opposed to a multitude of other *ad hoc* approaches that exist in the literature. The method of producing the BNs from data that we use is one that has proven to be scientifically acceptable in the machine learning community and good in practice [16, 23].

The remainder of the paper is organized as follows. In section 2 we briefly review the current literature on DataCubes and the prevalent current implementation, bitmaps, and also of Bayesian networks. In section 3 we present a simple introduction to Bayesian networks and methods of inducing their structure from data. In section 4 we describe our approach, and we show some experimental results in section 5. We conclude with a discussion of relevant issues and directions of future research in section 6.

2 Related Work

DataCubes were introduced in [8]. They may be used, in theory, to answer any query quickly (*e.g.* constant time for a table-lookup representation). In practice however they have proven exceedingly difficult to compute and store because of their inherently exponential nature. To solve this problem, several approaches have been proposed. [9] suggest materializing only a subset of views and propose a principled way of selecting which ones to prefer. Their system computes the query from those views at run time. Cubes containing only cells of some minimum support are suggested in [2] and a coarse-to-fine traversal is proposed that improves speed by condensing cells of less that the minimum support. Histogram-based approaches also exist [13], as well as approximations such as histogram compression using the DCT transform [17] or wavelets [24]. Perhaps closest to our approach is [1], which uses linear regression to model DataCubes. Bitmaps are relatively recent method for efficiently computing counts from highly compressed bitmapped information about the properties of records in the database. They are exact techniques. Unlike the DataCube and Bayesian networks, bitmaps do not maintain counts, but instead perform a pass over several bitmaps at runtime in order to answer an aggregate query [14, 3]. Query optimizers for bitmaps also exist [25].

There has not been much work on applying Bayesian networks to databases. An exception is [21], where possible causal relations from data are computed for purposes of data mining. Also, [6] used Bayesian networks for lossless data compression applied to relatively small datasets. Data mining research itself has been mostly focused on discovering association rules from data [18, 15], which can be viewed of as a special case of Bayesian network induction.

Bayesian network research on the other hand has flourished in the last decade, spurred mostly by Pearl's seminal book [19]. [10] contains a comprehensive overview of approaches to inference and structure induction. Restricted classes of Bayesian networks such as trees have been solved optimally [5] in the past. However, the general problem is NP-complete [4]. There exist two general approaches: the hill-climbing approach based on the MDL score [16, 23], the prevalent, more practical one which is used here, and the constraint-based approach. Constraint-based algorithms are covered in [22].

3 Bayesian Network Overview

In this section we present some background on Bayesian networks, emphasizing the points that relate to our current application of decision support systems and data mining. We also discuss methods to automatically compute their structure from data samples (database records) taken from a domain of interest. It should be noted that this is a difficult problem in its own right. In this paper we draw from years of research for the most practical and widely accepted solution for memory-resident databases, and propose a new algorithm to attack the problem in the context of very large databases that cannot fit in main memory.

Before we begin, we introduce some notation: we denote variables with capital letters X, Y etc.) and their values with lower-case letters (x, y etc.). Sets are shown in bold letters ($\mathbf{U}$, $\mathbf{D}$ etc.). The symbols used throughout the paper is shown in the table on the next page.

The attributes in the database are also referred to as "variables" throughout the paper, since they have a one-to-one correspondence to the variables that appear in the Bayesian network.

Table of Symbols	
$\mathbf{D}$	Main database
N	Number of records in main database *i.e.* $\lvert\mathbf{D}\rvert$
$\mathbf{U}$	Set of attributes: $\{X_1, \ldots, X_n\}$
n	Number of attributes *i.e.* $\lvert\mathbf{U}\rvert$
$(x_1, x_2, \ldots, x_n)_j$	Tuple j in database, $j = 1, \ldots, N$
$\mathbf{D}_i$	Database subset used to construct Bayesian network B_i
m	Size of each subset $\lvert\mathbf{D}_i\rvert$, $i = 1, \ldots, K$
K	Number of database subsets *i.e.* $\lceil N/m \rceil$
B_i	Bayesian network constructed from $\mathbf{D}_i$, equal to $\langle E_i, T_i \rangle$
$\mathbf{E}_\mathrm{i}$	Set of edges of graph of B_i
$\mathbf{T}_\mathrm{i}$	Set of probability tables of B_i

3.1 A Brief Introduction to Bayesian Networks

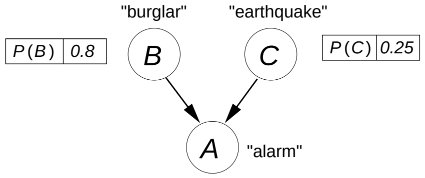
A Bayesian network (**BN**) is a graphical representation of a probability distribution function over a set of variables $\mathbf{U} = \{X_1, X_2, \ldots, X_n\}$. It consists of two parts: (a) the directed network structure, and (b) the conditional probability tables, one for each variable. The network structure is constrained to be acyclic. Undirected cycles are allowed *i.e.* cycles along which not all edges are pointed in the same way. Such structures represent alternative paths of influence between variables. The variables are typically discrete, although BNs with continuous variables are also possible.

As an example, a simple Bayesian network is shown in figure 1. It depicts three boolean variables, A ("home alarm goes off"), B ("burglar enters the house") and C ("earthquake occurs"). In this paper we will assume that all variables are binary, although this is not necessary and does not affect the generality of our approach. In the binary case, each conditional probability table records the probabilities of that variable takes the value "true" for each possible combination of values ("true" or "false") of its parents. The meaning of the BN in figure 1 is that A depends on B and A depends on C but B and C are independent.

In general, a probability distribution can be specified with a set of numbers whose size is exponential in $\lvert\mathbf{U}\rvert$, namely the entries in the joint probability distribution table. One can represent such a table by a completely connected BN in general, without any great benefit. However, when independencies exist in the domain, using a BN instead of the full joint probability table results in two major benefits:

1. **Storage savings**. These may be significant to the point where infeasibly large domains may be representable, provided that they exhibit a sufficient number of independencies among the variables of the domain. The savings are typically exponential because conditional independencies are common in practice.
2. **Clear and intuitive representation of independencies**. Given the graphical representation of a BN, it is easy to determine the variables on which a quantity of interest depends on statistically and which are irrelevant and under what conditions.

Edge omissions indicate the existence of conditional independencies among variables in the domain. As mentioned above, if all variables in the domain statistically depend on all others, then there is no storage advantage to using a BN, since the storage required for the specification of the network is exponential in $\lvert\mathbf{U}\rvert$. Fortunately, in practice this is not the norm, and in fact the most interesting domains for data mining are those that exhibit a considerable number of independencies.



$P(A\mid BC)$	0.2
$P(A\mid B\overline{C})$	0.6
$P(A\mid \overline{B}C)$	0.4
$P(A\mid \overline{BC})$	0.1

(a) Bayesian network

$\varnothing$	1000

A	435
$\overline{A}$	565

B	800
$\overline{B}$	200

C	250
$\overline{C}$	750

AB	400
$A\overline{B}$	35
$\overline{A}B$	400
$\overline{AB}$	165

$\overline{A}C$	60
$A\overline{C}$	375
$\overline{A}C$	190
$\overline{AC}$	375

BC	200
$B\overline{C}$	600
$\overline{B}C$	50
$\overline{BC}$	150

ABC	40
$AB\overline{C}$	360
$A\overline{B}C$	20
$A\overline{BC}$	15
$\overline{A}BC$	160
$\overline{A}B\overline{C}$	240
$\overline{AB}C$	30
$\overline{ABC}$	135

(b) Corresponding DataCube counts

Figure 1: (a) Example Bayesian network and (b) DataCube constructed from a database of 1,000 examples. The Bayesian network consumes less space in this example because B and C are independent.

In order to illustrate the storage space savings in this domain, we can look at figure 1(b). There we see the complete DataCube of the domain using a database that contains 1,000 examples. The numbers that have to be stored in the DataCube are 22 essential counters. The numbers necessary in the corresponding BN are 6 probability entries. We see that for this particular example this is certainly not a significant improvement, especially considering the overhead of specifying the parents of each node and using floating point numbers for the probability entries. However, for large networks with tens or hundreds of variables, the savings increases exponentially, if the corresponding network is sparse. For n attributes, the DataCube has to store 2^n tables of counts, with each table having size equal to the product of the cardinalities of the attributes they include (minus one). No full joint table for hundreds of variables containing either probabilities or counts could ever be stored for example using today's technology. However, such a domain can be succinctly represented by its joint probability distribution by taking into account the independencies that exist and using its Bayesian network instead. Such is the approach that we propose in this paper.

The independencies expressed by a Bayesian network can be easily read from its structure. In figure 1 for example, B ("burglar") and C ("earthquake") are independent in the absence of any knowledge about A ("alarm").[1] If they were not, then either edge $B \to C$ or $C \to B$ would have to have been included in the network.

3.2 Computation of the Bayesian Network Structure

The discussion of the previous section serves to establish the usefulness of modeling data domains using BNs. However, BNs are not as widely used as more traditional methods such as bitmaps for example, especially within the database community. We argue that the main reason for this, apart from the fact that the database and machine learning communities are mostly separate, lies in the computational difficulty in inducing models from data. Bayesian network induction from data is not an exception. However, we argue in this paper that the benefits are great, especially in domains such as decision support systems and data mining where they are a natural fit. In this paper we present an algorithm for computing and querying BNs constructed from very large databases that cannot fit in main memory, solving one of the main obstacles in adopting such a promising approach to many important problems in the database community.

The main difficulty in modeling with BNs lies in determining the structure of the graph. Once the graphical structure has been determined, the computation of the conditional probability tables is simple: a simple counting procedure of the records in the database that correspond to each entry would achieve a maximum likelihood estimate of the true table entries.

The problem of structure discovery from data is NP-complete in its generality [4]. In practice, there exist several potential solutions. One is to use a domain expert to specify the structure, but that is a slow and error-prone process, especially in large domains with many variables. In such domains, like market basket analysis, such a specification by an expert may be impossible or at least questionable. Therefore there is great benefit in both speed and reliability in automating this procedure as much as possible. Currently the most widely accepted method of BN structure discovery from data is described in [16, 23]. The method works by using heuristic search in the space of legal structures (*i.e.* structures without directed cycles) attempting to optimize an objective function, frequently referred to as *score*. The score that is frequently used in practice is the Minimum Description Length (MDL) of the data by the BN. We note here that this score is theoretically derived and not arbitrary. The heuristic search procedure frequently used is greedy hill-climbing, where each step consists of an addition, deletion or reversal of an edge in the current network that is under consideration during search. The search procedure can be initialized in a variety of ways, such as an empty, a completely connected or a randomly created initial network. The direct objective of this method is to produce a network that describes the data well so that its predictions (probability estimates) of the data records are as accurate as possible without at the same time producing a overly complex network (that is consuming a large amount of storage space).

The algorithm outlined in the previous paragraph is the one we used in this paper. It is implemented within the $BuildFromMemoryUsingData(\mathbf{D})$ routine that is shown below, adapted from [16, 23].

Procedure $B = BuildFromMemoryUsingData(\mathbf{D})$

1. $\mathbf{E} \leftarrow \emptyset$
2. $\mathbf{T} \leftarrow ProbabilityTables(E, \mathbf{D})$
3. $B \leftarrow \langle \mathbf{E}, \mathbf{T} \rangle$
4. $score \leftarrow -\infty$
5. do:
   (a) $maxscore \leftarrow score$
   (b) for each attribute pair (X, Y) do
   (c) for each $\mathbf{E}' \in \{\mathbf{E} \cup \{X \to Y\},$ $\mathbf{E} - \{X \to Y\},$ $\mathbf{E} - \{X \to Y\} \cup \{Y \to X\}\}$
   (d) $\mathbf{T}' \leftarrow ProbabilityTables(\mathbf{E}', \mathbf{D})$
   (e) $B' \leftarrow \langle \mathbf{E}', \mathbf{T}' \rangle$
   (f) $newscore \leftarrow ScoreFromData(B', \mathbf{D}))$
   (g) if $newscore > score$ then
   $B \leftarrow B'$
   $score \leftarrow newscore$
6. while $score > maxscore$
7. Return B

The procedure $ProbabilityTables()$ is typically a straightforward maximum-likelihood estimation of the probability entries from the database, which consists of counting the number of database records that fall into each table entry of each probability table in the BN.

The score that is used and the probability table computation are central points in our approach since they are the only places in any of our algorithms (preprocessing or querying) that the database $\mathbf{D}$ is accessed. The MDL score, mentioned above, is defined as follows:

$$ScoreFromData(B, \mathbf{D}) = -\sum_{i=1}^{N} p_i \log p_i - penalty(B, N)$$

$$\begin{aligned} p_i &= Pr((x_1, x_2, \ldots, x_n)_i, \mid B) \\ &= \prod_{j=1}^{n} Pr(X_j = (x_j)_i \mid \mathbf{Pa}_j, \mathbf{D}) \end{aligned}$$

$$penalty(B, N) = (|\mathbf{T}_i| / 2) \log N$$

where $\mathbf{Pa}_j$ is the set of parents of variable X_j in B, the conditional probability $Pr(X_j = (x_j)_i \mid \mathbf{Pa}_j, \mathbf{D})$ is computed by simple counting within the database $\mathbf{D}$, and $|\mathbf{T}_i|$

[1] It is less intuitive that this structure implies that B and C become dependent if the value of A is known. Although this makes a lot of sense, it is not within the scope of this paper and we will not discuss it further. For more details, please see [22].

is the number of necessary entries in all tables of B_i (the number of parameters of the model B_i). We see that the score has two components: one that describes how well the network B describes the data ($-\sum p_i \log p_i$) and one that penalizes B for being too large (see discussion in section 1 and [16, 23] for details). In section 4.2.1 we will describe how to compute this score, and in particular the first term, from a set of Bayesian networks that represent our database, instead of the records themselves. This is necessary when merging BNs representing portions of $\mathbf{D}$ into a single BN, without accessing the data (see below).

In section 4.2.1 we also show how to implement the $ProbabilityTables()$ procedure without accessing the database.

3.3 Using the BN for Probability Estimation: BN Inference

After generating a single BN for our database, we can use it to answer count queries. In order to do that, we need to estimate the probability (expected frequency) of the query using the BN, and multiply it with the number of records in the database (see section 4.2.2). *We do not need to access the database* for this.

The computation of this probability may be involved and in general cannot be simply read off the probabilities in the tables of the network. For example consider two variables X and Y that are very far apart but connected by a directed path. The probability of $X = 0$ and $Y = 1$ without knowledge of the value of any other variable in the network is not a simple function of the entries in the conditional probability tables of the BN. Rather, it requires a process called probabilistic inference.[2]

There exist several algorithms for inference. Two kinds of methods exist: approximate and exact. Approximate ones [11, 7, 20] are sample-based, and generate an artificial database of samples during the process of estimation (the generated samples are discarded immediately and only the count of those than matched the query is kept). Their main disadvantage is that they are slow and may need a great number of samples to estimate the probability of the query to a sufficient degree. For exact inference, the most popular method is the join-tree algorithm. The details of the algorithm are beyond the scope of this paper, please see [19, 12]. Its running time depends on the number of variables and the complexity of the BN, but in practice for typical BNs of a few tens of variables it runs in under a second. This is the method we use in this paper, contained in the $EstimateProbability()$ procedure that appears in section 4.2.2.

4 Proposed Method

4.1 Problem Description

The problem we are addressing is the following:

> **Problem**: we are given a database that does not fit in memory and a procedure $BuildFromMemoryUsingData(\mathbf{D})$ that is able to generate a BN from a memory-resident database.
>
> **Desired Solution**: a representation that can fit in memory and a procedure $EstimateCount()$ that uses it to compute the approximate answer to count queries that may specify an arbitrary number of attribute values.

The naive DataCube solution is to preprocess and store the counts for all possible such queries (see example in figure 1). However, this is infeasible for almost any realistic domain. Compressed bitmaps are one way of answering such queries exactly. However they may exceed the main memory size for very large databases. Also, since their perfect accuracy is not needed in the kind of applications we are addressing in this paper, it is reasonable to trade-off a small amount of accuracy in exchange for a much smaller representation that can fit in main memory, which in turn translates to a significant benefit in query performance. Sampling is one approximate technique that is however linear time ($O(N)$) in the database size, as are bitmaps.

In this paper we propose to represent the record counts in the database with a single Bayesian network created from the entire database $\mathbf{D}$. Our method is *constant time* ($O(1)$) in the size of the database. It consists of merging a number of BNs, B_i, each constructed from a partition $\mathbf{D}_i$ of the entire database into a single one, $\mathcal{B}$. Each B_i is created directly from $\mathbf{D}_i$ if it fits into main memory, or else by recursively splitting it, creating a network from each piece, and combining them in the same fashion that we combine the B_i's into $\mathcal{B}$. Each network B_i represents the joint probability distribution of partition $\mathbf{D}_i$. Since the B_i's are typically far smaller than the corresponding database partition $\mathbf{D}_i$, they can have the benefit of (1) simultaneously fitting into main memory and (2) tremendously speeding up the generation of the single network $\mathcal{B}$ since no disk access is required (we do not access the database during merging or at query time).

The answer to a query is computed by using $\mathcal{B}$ to compute the probability (support) of the query and multiplying it with with the number of records in $\mathbf{D}$ (denoted as N in the paper).

4.2 Proposed Algorithms

Before we present the algorithms, some notation (also see our table of symbols): the entire database as a set of records is $\mathbf{D}$, and we denote each partition that we use to construct a Bayesian network B_i from as $\mathbf{D}_i$. Therefore $\bigcup_{i=1}^{N} \mathbf{D}_i = \mathbf{D}$ and $\mathbf{D}_i \bigcap \mathbf{D}_j = \emptyset$, for $i \neq j$. We want each partition $\mathbf{D}_i$ to be large enough so as to be representative, but small enough so that it fits into the main memory and satisfies the time constraints for building the corresponding Bayesian network. In the next two sections we describe the algorithm to merge a number of Bayesian

[2] Which is a generalization of logical inference—given a BN, it computes the probability of the truth of a compound predicate (query) rather than a true/false value.

networks, each constructed from a database partition using the $BuildFromMemoryUsingData(\mathbf{D}_i)$ procedure, and the algorithm to compute the count that corresponds to a user query.

4.2.1 Algorithm for preprocessing the database: building and merging the BNs

The proposed procedure to build Bayesian network $\mathcal{B}$ from data stored on disk is as shown in Figure 2. Inside the procedure, the $BuildFromMemoryUsingData()$ procedure contains the implementation of the algorithm for finding the structure of a Bayesian network from data that was described in section 3.2. We note that the generation of each B_i can be done in parallel.

Having produced the networks $B_i, i = 1, \ldots, K$, we combine them into a single one, $\mathcal{B}$, using the following procedure:

Procedure $\mathcal{B} = RecursivelyMerge(B_1, \ldots, B_K)$:

If $B_1, B_2, \ldots, B_K$ simultaneously fit in main memory then:
 $\mathcal{B} = BuildFromMemoryUsingBNs(B_1, \ldots, B_K)$
else:
 $\tilde{B}_1 = RecursivelyMerge(B_1, \ldots, B_{\lfloor \frac{K}{2} \rfloor})$.
 $\tilde{B}_2 = RecursivelyMerge(B_{\lfloor \frac{K}{2} \rfloor + 1}, \ldots B_K)$.
 $\mathcal{B} = RecursivelyMerge(\tilde{B}_1, \tilde{B}_2)$.

The $BuildFromMemoryUsingBNs(B_1, \ldots, B_K)$ procedure is the only remaining one that needs to be defined. It is exactly the same as the $BuildFromMemoryUsingData(\mathbf{D})$ one (see section 3.2), with the exception that the score is now computed from the BNs ($ScoreFromBNs()$ procedure) that are its arguments instead of the database ($ScoreFromData()$ procedure):

$$ScoreFromBNs(\tilde{B}, \overbrace{B_1, \ldots, B_K}^{\text{representing } \mathbf{D}}) = \\ \sum_{t \in Tables(\tilde{B})} \log Pr(t \mid \tilde{B}) \\ \times \left[(1/K) \sum_{k=1}^{K} EstimateProbability(t, B_k) \right] \\ - penalty(\tilde{B}, N).$$

In the above formula the outer sum goes over all table entries t in $\tilde{B}$. Each such table entry corresponds to a configuration of variable assignments (for the node and the parents of the node that it is attached to) and "don't cares" (for the remaining variables in the domain)—see figure 1 for example. The inner equally-weighted sum is simply an average over all networks $B_i, i = 1, \ldots, K$ of the probability of that configuration. $Pr(t \mid \tilde{B})$ is the probability of configuration t in $\tilde{B}$, and can be read directly off the corresponding table entry of $\tilde{B}$.

The computation of the probability tables by the $ProbabilityTables()$ procedure is also done from the B_i's without accessing the database; it is making use of the $EstimateProbability()$ procedure:

$$\forall t \in Tables(\tilde{B}) \\ Pr(t) = (1/K) \sum_{k=1}^{K} EstimateProbability(t, B_k)$$

Since the database access is $O(N)$ during the $BuildFromDisk(\mathbf{D})$ procedure, the number networks at the base of the recursion is $K = N/m = O(N)$, and since accessing a BN does not depend on the database size, it is easy to make the following observation:

> **Observation**: the entire $BuildFromDisk()$ algorithm is $O(N)$ time (linear in the size of the original database) and thus scalable. Moreover, it is parallelizable, with a straightforward parallel implementation.

This observation is supported by the experimental results (section 5, figure 7).

4.2.2 Algorithm for answering a count query from a Bayesian network

To estimate approximate counts for query Q from the Bayesian network $\mathcal{B}$ that is is the output of the $BuildFromDisk()$ procedure, we use the $EstimateCount()$ procedure, shown below:

Procedure $\hat{N} = EstimateCount(Q, \mathcal{B})$:
 $\hat{N} = N \times EstimateProbability(Q, \mathcal{B})$.

The procedure $EstimateProbability()$ can be any inference method developed in the Bayesian network literature. In our implementation we use the join-tree algorithm, which is a well-known exact algorithm for computing the probability of the query. $EstimateProbability()$ returns the probability of query Q according to the probability distribution represented by $\mathcal{B}$. Since $\mathcal{B}$ is a representative of the N records contained in $\mathbf{D}$, $N \times EstimateProbability(Q, \mathcal{B})$ is an estimate of the number of records within $\mathbf{D}$ for which Q evaluates to "true."

Since the $EstimateCount(Q, \mathcal{B})$ algorithm does not access the database, under our assumptions we can make the following observation.

> **Observation**: the $EstimateCount(Q, \mathcal{B})$ procedure is $O(1)$ time in the size of the database.

This observation is also supported by our experimental results (section 5, figure 5).

5 Experimental Results

We experimentally tested our approach on real and synthetic data. The real data consists of customer information data, obtained from a large anonymous retailer.[3] It consists of over 3 million customer transactions (3,261,809)

[3]For confidentiality reasons we cannot reveal the name the retailer nor the products involved.

Procedure $(B_1, B_2, \ldots, B_K) = BuildFromDisk(\mathbf{D})$:

1. Partition the database $\mathbf{D}$ into N equal partitions $\mathbf{D}_i$, $i = 1, \ldots, K$ so that each fits in main memory. Let $m = |\mathbf{D}_i|$, for all i.
2. For each $i = 1, \ldots, K$ do the following:
   (a) Read $\mathbf{D}_i$ into memory.
   (b) Build Bayesian net B_i from $\mathbf{D}_i$: $B_i = BuildFromMemoryUsingData(\mathbf{D}_i)$.
3. Merge the networks B_i into a single one: $\mathcal{B} = RecursivelyMerge(B_1, B_2, \ldots, B_K)$.

Figure 2: Algorithm for preprocessing the database.

containing information on whether the customer purchased any of the 20 most popular items in the store. The data represents one week of activity and its concise representation occupies around 8 MB. This size coincides with the size of its uncompressed bitmap. Although this database is not large in size, we use it in order to obtain performance results on the compression ratio we can obtain on real-world data.

In order to assess the scalability of our system, we needed larger sets that were not available at the time of our evaluation. For this reason we used synthetic data for our scalability study. The synthetic data we used were produced by a program available from IBM's QUEST site.[4] The generation program produces a specified number of randomly generated association rules involving a number of attributes (their number is also randomly distributed around a user-specified mean), and then generates market-basket data whose statistical behavior conforms to those rules. We produced a database of approximately 100 thousand and 1, 10, 100 and 196 million records from a store inventory of 5,000 items (products) using 10,000 customer patterns having an average length of 4. (Each customer pattern corresponds to an "association rule.") The average transaction length was 10 items. As in our real database, we used the 20 most frequently used items.

From both real and synthetic databases we then constructed a number of Bayesian networks from that data in order to model their joint probability distribution. We split the data randomly in a number of subsets $\mathbf{D}_i$, each containing at most $m = 100,000$ records. We then used each subset $\mathbf{D}_i$ to construct the corresponding Bayesian network B_i.

Our experiments evaluate our approach with respect to the following dimensions:

1. Query count error.
2. Space to store models and effective compression of the database.
3. Time to answer a query.
4. Build time and scalability.
5. Visualization of the dependencies in the database.

Because the number of possible queries grows exponentially with the number of variables that are allowed to be involved in it, we were not able to perform all possible queries of any sizeable length. Instead we generated 10,000

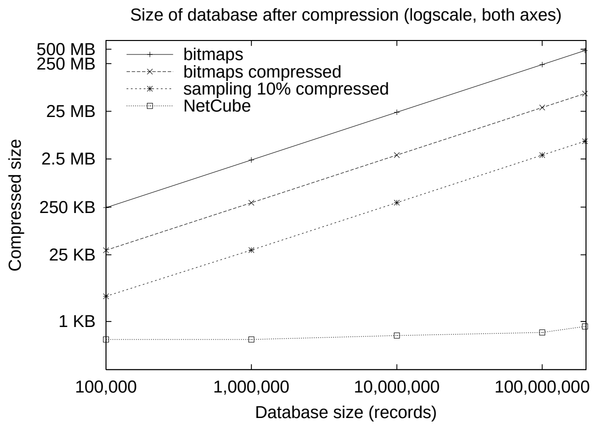


Figure 4: Comparison of the size of the compressed database size using bitmaps, sampling by 10% and NetCubes. The difference between `gzip` and `bzip2` is small (see table 3), so only the best of the two (`bzip2`) is used here.

random queries of length up to 5 variables and used them to assess the query error. Each query may test for the presence **or absence** of any particular item in a transaction, from the 20 most frequently purchased items. For example one such query may be "what is the number of transactions in the database in which a customer purchased milk and orange juice but not bread?"

Compression

In this set of experiments we compare the size of our representation to that of compressed bitmaps and sampling by 10%, also compressed. Compressing the bitmaps of each of our databases produced approximate 7:1 compression ratio for the synthetic Quest databases and 3.8:1 for the real-world data. Compressing the sampled database predictably produces linear compression with respect to compressed bitmaps. In stark contrast, the NetCube approach typically produced compression ratios of 500:1 or more for synthetic data and 1800:1 or more for real data. The compression ratios and BN sizes are shown in table 3 and are also plotted in figure 4. The price for such a high compression performance is the fact that it is lossy. However, if the application can tolerate errors of the order of 5%, then it may be the method of choice for the data analyst, since sampling by 10% achieves much lower compression ratios

[4] `http://www.almaden.ibm.com/cs/quest/`

Database	Records	Bitmap size (bytes)	Compression ratios (*before*:*after*)			
			`gzip`	`bzip2`	Sampling	NetCube
Quest	100,000	250,000	7:1	7.8:1	72:1	**581:1** (**430 bytes**)
	1,000,000	2,500,000	7.1:1	7.9:1	77:1	**5814:1** (**430 bytes**)
	10,000,000	25,000,000	7.1:1	7.9:1	79:1	**48170:1** (**519 bytes**)
	100,000,000	250,000,000	7.1:1	8:1	79:1	**414594:1** (**603 bytes**)
	196,896,433	492,241,100	7.1:1	8:1	80:1	**610721:1** (**806 bytes**)
Anonymous retailer	3,261,809	8,154,540	3.8:1	3.8:1	37:1	**1889:1** (**4317 bytes**)

Figure 3: Comparison of compression ratios for various databases used for the experiments. The first rows correspond to the Quest-generated databases while the last one corresponds to real data obtained from an anonymous retailer. The sampling figures refer to 10% sampling and after `bzip2` compression. For the NetCube, the trend of compression ratios that are increasing with database size is due to increasing benefits from using an approximately fixed-sized probabilistic model of a domain in place of data drawn from it.

with similar error; moreover, the query time for sampling is linear in the size of the database while NetCube queries are approximately constant.

Note that the network produced from real data, corresponding to one week of transactions, occupies only 4 KB. If are allowed to make the conservative assumption that the network from any given week is 10 times this one (40 KB), and the assumption that doubling the database size doubles the size of the resulting network (for which our experiments have no support of, and in fact indicate that it might not grow at that rate but a much smaller one), then our approach makes it possible to fit **20 billion** transactions in the memory of a regular workstation with 256 MB of main memory, corresponding to **more than 100 years of transactions** at this rate, effectively spanning the lifetime of most businesses.

Query time

We used a typical workstation with 256 MB of physical memory for our query time experiments. Running our set of queries on the bitmaps we noticed a slowdown for the larger Quest databases whose bitmap cannot fit into main memory. This happens because the bitmap system had to use part of the virtual memory system which resides on the disk. An important observation we can make here is that although bitmap compression will temporarily alleviate this problem, a database of more than 7 times our largest one would again force the bitmap method into the same thrashing behavior (note the compression ratio 7:1 for bitmaps in table 3). A database of such a problematic size would not be atypical in today's real-world problems.

Even without the thrashing problem of bitmaps however, we see that the query times for sampling increase (linearly) with database size, as expected. This also shows in figure 5. In contrast, using a set of BNs to answer queries is approximately constant. That proves that our method can be an invaluable tool that remains practical for extremely large problems.

Query error

In figure 6 we show our assessment of the query error using our set of 10,000 random queries containing up to 5 variables. Because relative error becomes artificially large for

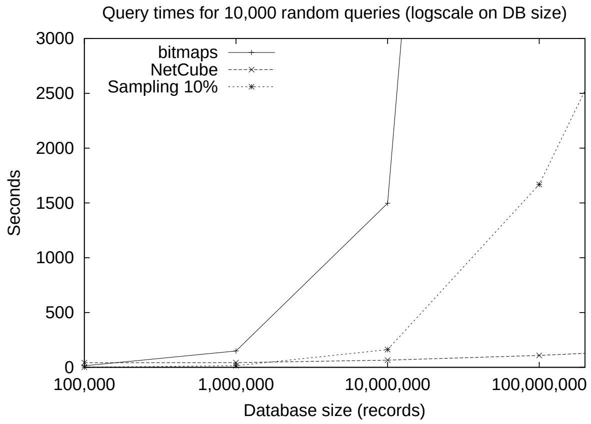


Figure 5: The query time for bitmaps increases linearly or superlinearly (thrashing) with the database size. Here 10,000 queries were run on a workstation with 256 MB of main memory. The NetCube query time is constant and under a second per query due to its approximately constant representation size.

queries of very little support even when the count difference is not very large, we used queries that had at support 1% or more. Apart from artificially weighing the error rate, queries of very small support are arguably "uninteresting" and can also be also due to spurious factors. Such treatment is consistent with other approaches in the literature (*e.g.* [2]).

In figure 6 we can see that the average relative error of NetCubes is very small, well under 5%. Those queries that happen to have larger than 5% error are typically those with small support, in the range of 1%. We also see that only sampling at 10% has average error levels comparable to NetCubes. That is the reason why in our evaluation we display figures for 10% sampling only.

Build time

As mentioned above, we generate a BN for each database piece of 100,000 records, *i.e.* $m = 100,000$ in our implementation. As we can see in figure 7, this makes our method linear on the database size, and thus scalable. Each database piece can be processed in parallel, and the merg-

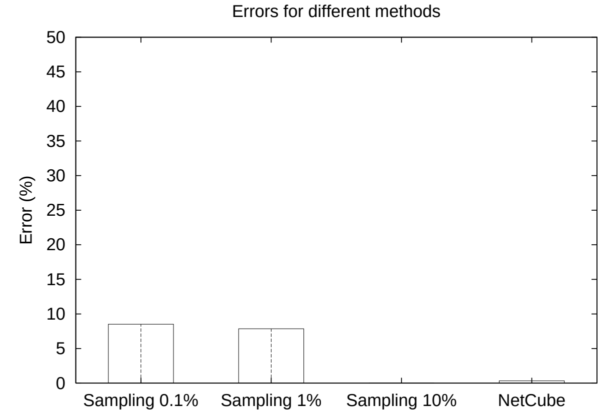


Figure 6: Average relative error rate for 10,000 random queries of up to 5 variables, having at least support of 1% of the size of the original database. The queries are done on the largest of the QUEST databases, containing 193 million records.

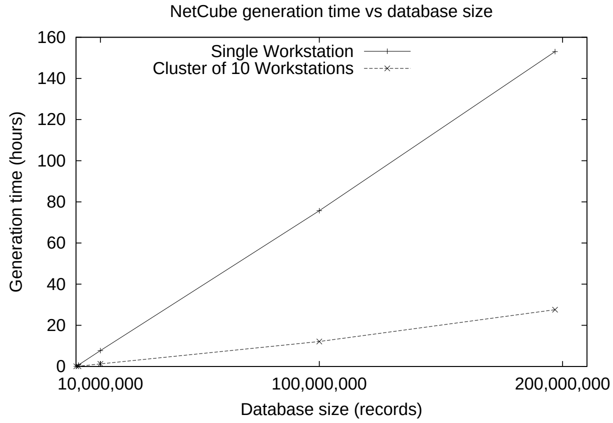


Figure 7: Build time for the set of multiple BNs increases linearly with the number of records in the database. Moreover, parallelization over a number of workstation scales the build time down linearly. Each partition of the database contains 100,000 records and can be processed independently of others.

ing of the BNs can also be done in parallel across the same recursion depth. Thus our method is parallelizable in a straightforward manner. Parallelization over a cluster of workstations scales linearly, making the generation of a database of 200 million transactions a matter of hours on a modest cluster of 10 workstations, as shown in figure 7.

We note here that our attempts to create a single BN by using the straightforward $BuildFromMemoryUsingData()$ algorithm on the entire database were unsuccessful for very large problems of size 100 million records or more; the algorithm **did not terminate while producing the network after 4 days and had to be manually aborted**. This clearly underscores the usefulness and indeed the absolute necessity of using our recursive combination procedure ($BuildFromDisk()$ procedure) for any kind of practical application that involves very large databases.

Visualization

In figure 8 we show a BN produced from real data corresponding to a week of activity of the 20 most frequently purchased items at a large anonymous retailer. The advantage of the graphical representation of the BN that our approach generates is that it can be used to clearly depict variables that are the most influential to the ones that the analyst might be examining. Moreover, the conditional probability tables will give our analyst the exact nature and strength of these influences. Therefore our approach fits very well in the data mining procedure and can save the analyst large amounts of time that would be otherwise spent on exploration, drill-down analysis etc. of the customer database.

6 Discussion and Extensions

In this paper we use only binary variables. However, the concepts and implementation easily extend to multi-valued discrete data easily. NetCubes can also handle continuous variables after bucketization *e.g.* "salary" could become a discrete variable taking values "low" ($\leq 10,000$), "medium" ($\geq 10,000$ and $\leq 100,000$) or "high" ($\geq 100,000$).

A subject of future research is the extension of the current system to the estimation of additional aggregate functions of the DataCube operator, in addition to counts. For example, knowing the probability distribution of a multi-valued attribute enables us to quickly estimate its average value. Other quantities such the minimum and maximum values can be read directly from representation of the Bayesian network.

The approach presented here lends itself easily to non-stationary distributions. Assume for example that new data are incorporated in the database periodically, *e.g.* a supermarket may append transaction data to the database at the end of each day. A data analyst may be interested in certain quantities on a per-day basis. In that case the solution is easy: we can compute one Bayesian network for each particular day only. That network can answer queries for that day. More often it is more useful to examine the behavior over broader time periods. The same approach will work for that purpose: a query concerning several days, not necessarily consecutive, can be made to the corresponding (single-day) networks covering the time period of interest. The resulting counts can then be simply summed to obtain a count estimate for the entire time period.

7 Conclusions—Contributions

In this paper we propose a *paradigm shift* in the approximate computation of count DataCubes: we propose to use a model of the data instead of the data themselves. Our approach, NetCube, uses the proven technology of Bayesian networks to obtain the key advantage of large storage savings in situations where only approximate an-

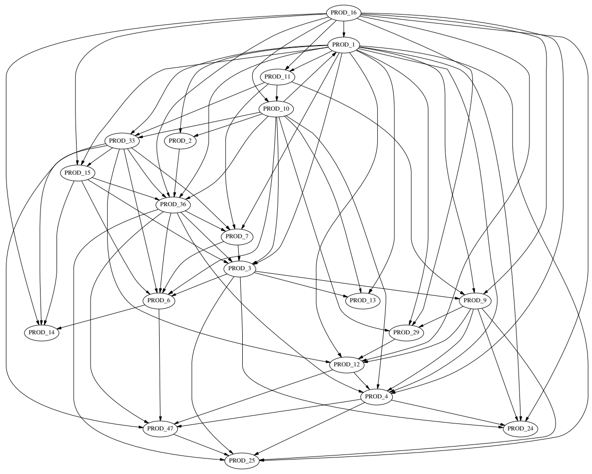


Figure 8: Bayesian network produced from real data obtained from a large anonymous retailer. The database contains 3,261,809 records corresponding to a week of customer transactions on the 20 most frequently purchased items. The network occupies 4317 bytes on disk. For confidentiality reasons, we have anonymized the names of the products that are displayed in the graph.

swers are needed. This makes feasible the computation of DataCubes for databases that were previously problematic using state-of-the-art methods such as bitmaps.

A size-error comparison of our NetCube method versus competing methods is shown in figure 9. In summary, the advantages of the method are:

- *Small space*: the resulting BN takes up a tiny fraction of the space that the original data that are queried upon. We produced **greater than 1800:1 compression ratios on real data**.
- *Scalability*: we can handle arbitrarily large databases; the method's preprocessing time scales **linearly** with the size of the database. Moreover, it is parallelizable with a straightforward parallel implementation.
- *Fast query time*: the method can answer arbitrary queries in a short time (typically under a second). The query time is **constant** with respect to the database size.
- *Good accuracy*: we obtained less than 5% average relative error on a large number of queries of support of 1% or more.
- *Suitability to data mining*: the representation that is used by the algorithm, namely Bayesian networks, are an excellent method for visually eliciting the most relevant causes of a quantity of interest and are a natural method to support data mining.

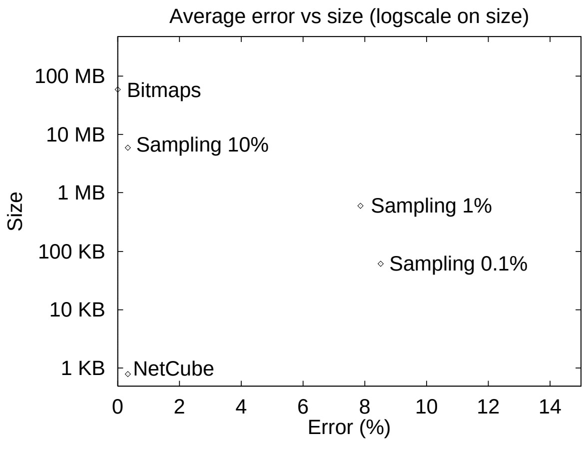


Figure 9: Comparison of NetCubes with bitmaps and sampling: representation size versus average query error.

References

[1] Daniel Barbará. Quasi-cubes: A space-efficient way to support approximate multidimensional databases. Technical report, ISE Dept., George Mason University, 1998.

[2] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and Iceberg CUBEs. In *Proceedings of the 25th VLDB Conference*, pages 359–370, 1999.

[3] C.-Y. Chan and Y. Ioannidis. Hierarchical cubes for range-sum queries. In *Proceedings of the 25th VLDB Conference*, pages 675–686, 1999.

[4] D. M. Chickering. Learning bayesian networks is NP-complete. In *Proceedings of AI and Statistics*, 1995.

[5] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14:462–467, 1968.

[6] S. Davies and A. Moore. Bayesian networks for lossless dataset compression. In *Conference on Knowledge Discovery in Databases (KDD)*, 1999.

[7] R. Fung and K.C. Chang. Weighting and integrating evidence for stochastic simulation in Bayesian networks. In L.N. Kanal, T.S. Levitt, and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence*. Elsevier Science Publishers B.V. (North-Holland), 1989.

[8] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data Cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In *International Conference on Data Engineering*, 1996.

[9] V. Harinarayan, A. Rajaraman, and J.D. Ullman. Implementing data cubes efficiently. In *SIGMOD*, pages 205–216, 1996.

[10] D. Heckerman. A tutorial on learning bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, Advanced Technology Division, March 1995.

[11] M. Henrion. Propagation of uncertainty by probabilistic logic sampling in Bayes' networks. In J. F. Lemmer and L. N. Kanal, editors, *Uncertainty in Artificial Intelligence 2*. Elsevier Science Publishers B.V. (North-Holland), 1988.

[12] C. Huang and A. Darwiche. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, 11:1–158, 1994.

[13] Y. Ioannidis and V. Poosala. Histogram-based approximation of set-valued query answers. In *Proceedings of the 25th VLDB Conference*, pages 174–185, 1999.

[14] T. Johnson. Performance measurements of compressed bitmap indices. In *Proceedings of the 25th VLDB Conference*, pages 278–289, 1999.

[15] R.J. Bayardo Jr. and R. Agrawal. Mining the most interesting rules. In *Proc. of the Fifth ACM SIGKDD International conference on Knowledge and Discovery*, pages 145–154, 1999.

[16] W. Lam and F. Bacchus. Learning bayesian belief networks: an approach based on the MDL principle. *Computational Intelligence*, 10:269–293, 1994.

[17] J.-H. Lee, D.-H. Kim, and C.-W. Chung. Multi-dimensional selectivity estimation using compressed histogram information. In *SIGMOD*, pages 205–214, 1999.

[18] N. Megiddo and R. Srikant. Discovering predictive association rules. In *Proc. of the 4th Int'l Conference on Knowledge Discovery in Databases and Data Mining*, New York, USA, August 1998.

[19] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, Revised Second Printing, 1997.

[20] R.D. Schachter and M.A. Peot. Simulation approaches to general probabilistic inference on belief networks. In L.N. Kanal, T.S. Levitt, and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence*. Elsevier Science Publishers B.V. (North-Holland), 1989.

[21] C. Silverstein, S. Brin, R. Motwani, and J. Ullman. Scalable techniques for mining causal structures. In *Proceedings of the 24th VLDB Conference*, pages 594–605, 1998.

[22] P. Spirtes, G. Glymour, and R. Scheines. *Causation, Prediction, and Search*. Springer-Verlag, New York, 1993.

[23] J. Suzuki. Learning Bayesian belief networks based on the MDL principle: an efficient algorithm using the branch and bound technique. In *Proceedings of the International Conference on Machine Learning*, Bally, Italy, 1996.

[24] J. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *SIGMOD*, pages 913–204, 1999.

[25] M.-C. Wu. Query optimization for selections using bitmaps. In *Proceedings of the 25th VLDB Conference*, pages 227–238, 1999.