Efficient Filtering of Large Dataset —— a User-Centric Paradigm

***Yi Xia*[*], *Wei Wang*[†], *Jiong Yang*[‡], *Philip Yu*[§], and *Richard Muntz*[¶]**

Abstract

In this paper, we investigate in the problem of efficient filtering of a large dataset according to some user specified criterion. A novel paradigm is introduced to allow users to easily integrate their prior knowledge in the mining process and to customize the mining result. The Bayesian Network is chosen as the model to represent user's knowledge due to its rich expressive power, compact representation, and intuitive semantics. However, one difficulty that prevents the Bayesian network model from being applied to large scale databases in a brute-force way is the generally high computational complexity of the inference algorithms. To tackle this challenge, we propose three techniques, namely *network pruning of conditional probability*, *computation reuse by tuple reordering*, and *early termination*, to speed up the process dramatically. It has been demonstrated that our proposed algorithm can save as much as 95% of the execution time empirically.

1 Introduction

Data mining techniques have been widely used to find potentially useful, implicit information in data. The mined information, once proved to be useful, is used as

[*]UCLA CS department, email address xiayi@cs.ucla.edu
[†]IBM T. J. Watson research, email address ww1@us.ibm.com
[‡]IBM T. J. Watson research, email address jiyang@us.ibm.com
[§]IBM T. J. Watson research, email address psyu@us.ibm.com
[¶]UCLA CS department, email address muntz@cs.ucla.edu

knowledge to guide further exploration of the data and to design and conduct many other tasks. Nowadays, the user's role in the mining process becomes increasingly important, and it has been observed that allowing the user to participate in the mining process cannot only provide more power to the user to express his (or her) interests and/or domain knowledge, but also often lead to a much more efficient process centered on the user's interests.

This paper aims at solving the following data mining problem. From a dataset of M attributes and N tuples[1], we want to efficiently identify the set of tuples satisfying a certain probabilistic criterion (typically defined on an unmaterialized attribute referred to as the *query variable*) specified by the user, and incorporating the user's prior knowledge on correlations (e.g., causal effects) among all attributes. There are many applications of this problem, one of which is target mailing. Merchants want to send product booklets to their potential customers. Considering the cost of booklets, they only want to send booklets to those for whom the possibility of purchasing the products is higher than some threshold. The question is, given some information about the customers, how to identify those potentially valuable customers.

The problem introduced above highlights the notion of user-centric paradigm. As part of the query input, the user can specify his knowledge (or belief) on correlations among attributes. Different users may have different understanding or emphasis on the correlations between attributes and may have different query criterion (in terms of different query variables and/or different probability ranges). The results reflect not only the characteristics of the data but also the user input knowledge. In this paper, we will not elaborate on how accurately the input knowledge models the real world, but rather focus on how to efficiently identify the set of tuples that satisfy the user specified criterion from a large database given the input knowledge. Sometimes, users may want to input pathological knowledge to simulate a virtual scenario for predictive analysis. A user-centric paradigm would provide users with the ability to inject their knowledge (either objective or subjective) to guide the mining process. There are two important issues towards this end: (1) how to represent user's input knowledge, and (2) how to efficiently determine the set of tuples satisfying the given probabilistic criterion.

To address the first issue, we adopt the Bayesian network as the means to represent the input knowledge. Bayesian networks have proved to be a powerful yet concise model to represent potential causality relationships among a set of variables and support probabilities inferencing. Its applications span education systems to medical systems [6, 7], weather forecasting [2] and market prediction [1]. A Bayesian network can be either learned from a training dataset or constructed according to a user's prior knowledge, or a combination of both. Users can integrate their knowledge or understanding of the world into the Bayesian network and feed it to the dataset as input. This provides a way for users to participate in the data mining process actively and to customize the outcome by encoding or emphasizing their knowledge/interest in the Bayesian network.

[1]Each tuple corresponds to an entity/object in the real world. It is possible (and indeed occurs quite often) that each tuple only possesses values of a subset of attributes due to the nature of the application or potential experimental errors or delays.

The second issue mentioned previously is performance. Unlike most previous work on Bayesian networks which focused on performing inference for a single tuple, our emphasis is on efficient inference for a large dataset applied to a given Bayesian network. One difficulty prevents the Bayesian network model from being applied in a brute-force way to large scale database is the generally high computational complexity of the inference algorithms. To tackle this challenge, we propose three techniques, namely *network pruning of conditional probability*, *computation reuse by tuple reordering* and *early termination*, and develop an approach based on the classical jointree algorithm for probabilities inferencing on Baysian networks [9, 8].

The remainder of this paper is organized as follows. Section 2 gives a brief overview of recent advances in inference algorithms for Bayesian networks. Section 3 formally defines the problem. Section 4 gives a full description of our algorithm. Experimental results are presented in Section 5. Finally, conclusions are drawn in Section 6.

2 Related Work

Bayesian inference algorithms have matured during the last 20 years. The polytree algorithm [11] is the simplest algorithm that only applies to a singly-connected network. To overcome its limitation in real applications, several algorithms have been proposed for multi-connected networks, which include but are not limited to cutset conditioning [11], jointree(clustering) [9, 8], and bucket-elimination [4]. Cutset conditioning parses the multi-connected network into a set of singly-connected networks and applies the polytree algorithm for each singly-connected network. The jointree algorithm derives a tree from the original network and inferences are based on the constructed tree. Bucket-elimination ranks the variables(nodes) in the Bayesian Network(BN) and associates each variable with a bucket. Each conditional probability table(CPT) in the BN is put into the bucket associated with its highest ranked variable. Inference proceeds by eliminating the buckets in the order their associated variables are ranked. All of these algorithms are exact inference algorithms. For very large and complex BNs, where exact algorithms become impratical in time and space, approximate algorithms are developed. One of them is Gibbs Sampling [10]. In this method, starting from an instantiation of all variables in the network that is consistent with the evidence, a large sample of instantiations are generated by simulating the flow of impact in the network. The generated sample is used to estimate the expectation of the requested probability. Mini-bucket elimination [5] is another approximate algorithm. It is a variant of the bucket elimination algorithm. One of its advantages is that it provides the flexibility of trading off accuracy and time/space.

All of above algorithms focus on the inference for a single tuple, while no research has been conducted on efficient inference for a large dataset. Our contribution in this paper is to reduce the average cost per tuple in the inference by introducing three new techniques. Some preliminary results on network pruning [3] and computation reuse [8] have been discussed in literature. There were not designed specifically for the conditional probability and did not consider reordering tuples to achieve an optimal result. In this paper, we extend these ideas to a deeper level.

3 Problem Description

Let $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$ be a set of attributes(variables) defined over multivalued domains $D_1, D_2, \ldots, D_n$. We use $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ to represent an instantiation of $\mathbf{X}$, where, $x_i \in D_i \cup \{unknown\}$, $i = 1, \ldots, n$. $\mathbf{x}$ is called a tuple defined over $\mathbf{X}$. We define $\mathcal{D}$ as a dataset composed of tuples over $\mathbf{X}$.

Let $\mathcal{B} = \{\mathcal{G}, \mathcal{T}\}$ be a Bayesian network, here $\mathcal{G} = \{\mathbf{V}, \mathbf{E}\}$ represents a directed acyclic graph with nodes $\mathbf{V}$ and edges $\mathbf{E}$. $\mathcal{T}$ represents the set of conditional probability tables(CPTs) associated with each node in $\mathbf{V}$. We refer to $\mathcal{B}$ as a *Bayesian network related to a dataset* $\mathcal{D}$ if each attribute in $\mathcal{D}$ has a corresponding node in $\mathcal{B}$. That is, the attributes in $\mathcal{D}$ correspond to a subset of nodes in $\mathcal{B}$. In the following discussion, we will not discriminate between the attributes in $\mathcal{D}$ and the nodes in $\mathcal{B}$, but mix the use of V and X to represent either a node in $\mathcal{B}$ or an attribute in $\mathcal{D}$ according to the context.

A criterion $\mathcal{C}(q, \mathbf{x}) : Pr(Q = q|\mathbf{X} = \mathbf{x}) > c\%$ is defined over the dataset $\mathcal{D}$ and is evaluated for each tuple $\mathbf{x}$ in $\mathcal{D}$ based on $\mathcal{B}$ as prior knowledge. Here, Q is called the *query variable* and $Q \in \mathbf{V} - \mathbf{X}$; $\mathbf{X}$ are called *evidence variables*, and their instantiation $\mathbf{x}$ serves as evidence/conditions in the conditional probability $Pr(Q = q|\mathbf{X} = \mathbf{x})$; $c\%$ is the probability threshold.

Our problem can be formally described as follows. The inputs include the dataset $\mathcal{D}$, the Bayesian network $\mathcal{B}$, and the probabilistic criterion $\mathcal{C}$. The output is a subset of tuples in $\mathcal{D}$: $\{\mathbf{x}|\mathbf{x} \in \mathcal{D}$ and $\mathcal{C}(\mathbf{x}) = true\}$. The evaluation of $\mathcal{C}(\mathbf{x})$ is based on the knowledge of $\mathcal{B}$.

As is discussed in Section 1, Bayesian network encodes a user's knowledge of the attributes in the related dataset. Different users may provide different Bayesian networks for the same dataset and lead to different assertions for the same tuple on the same criterion. The dataset could be incomplete in the sense that some tuples may contain *unknown* values, which is naturally accomodated in Baysian inferencing.

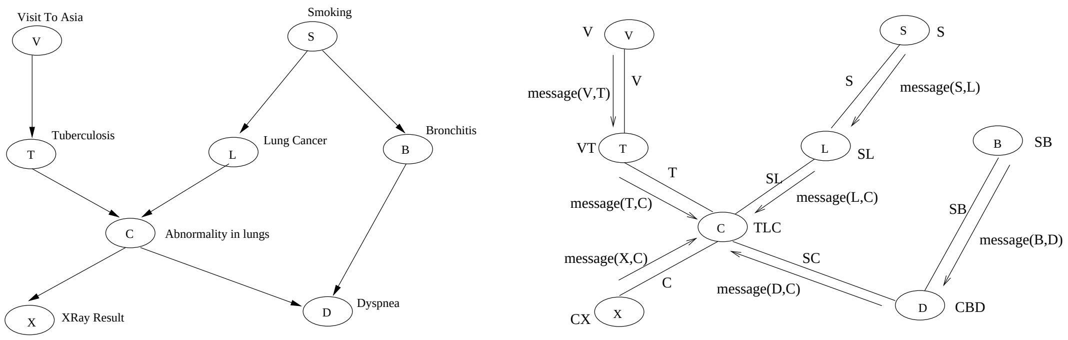
Let's look at an example. Table 1 lists a small portion of the dataset that contains patient information. Each tuple in the table represents a patient. A user provided BN is given in Figure 1(a), with the CPT of each node defined in Table 2. One application is to find the set of patients whose probability of having abnormality in lungs is greater than 40%. We will use this example throughout this paper.

4 Description of the algorithm

The example in the previous section can be solved by directly applying a Bayesian inference algorithm on each tuple in the dataset and computing the conditional probability $Pr(C = c|v, s, x, d)$, where (v, s, x, d) is a tuple in the dataset $\mathcal{D}$ with variables V, S, X, D. By comparing the result with the threshold $c\%$ (e.g., 40%), we can conclude whether a patient satisfies the probabilistic criterion.

Patient	Visit To Asia	X-Ray Result	Smoking	Dyspnea
Patient 0	True	Abnormal	False	Present
Patient 1	False	Normal	False	Present
Patient 2	False	Normal	True	Present
Patient 3	True	Abnormal	True	Present
...	...	...	...	...

Table 1. *a sample dataset containing patient's information*



(a) An example of Bayesian network

(b) Jointree of the Bayesian network

Figure 1. *A medical example: Chest clinic*

As the jointree algorithm is the most popular inference algorithm for Bayesian network, we adopt it as the basic foundation of our proposed algorithm. The jointree algorithm makes inference on a second structure, called the jointree, which is derived from the original network[2]. Before discussing further, we introduce several definitions related to jointree.

Definition 4.1. Basic jointree*: A basic jointree for a DAG* $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ *is a labeled tree* $\mathcal{T} = (\mathbf{V}, \mathbf{E}^{'}, \mathbf{F})$*, where*

- $\mathbf{V}$ *and* $\mathbf{E}$ *are the set of nodes and edges in* $\mathcal{G}$ *respectively.* $\mathcal{T}$ *has the same set of nodes as* $\mathcal{G}$*.* $\mathbf{E}^{'}$ *is a subset of* $\mathbf{E}$ *such that all nodes in* $\mathbf{V}$ *are singly-connected by* $\mathbf{E}^{'}$ *regardless of direction;*
- *Each node* i *in* $\mathcal{T}$ *is labeled with* F_i $(F_i \in \mathbf{F})$*, where* F_i *is the family of the node* i *in* $\mathcal{G}$ *and is defined as the union of the node* i *with all its parent nodes* $\mathbf{Pa}_i$ *in* $\mathcal{G}$*.*

For any Bayesian Network, each node i in $\mathcal{T}$ is associated with a conditional

[2]There are heuristic algorithms to construct a jointree of good quality. However, we just consider a simple way of constructing basic jointrees, which is introduced in Darwiche's paper [3].

Visit To Asia	
V	Pr_V
False	0.99
True	0.01

Smoking	
S	Pr_S
False	0.50
True	0.50

X-Ray Result		
X	C	$Pr_{X\|C}$
Normal	False	0.95
Abnormal	False	0.05
Normal	True	0.02
Abnormal	True	0.98

Lung Cancer		
L	S	$Pr_{L\|S}$
False	False	0.99
True	False	0.01
False	True	0.90
True	True	0.10

Table 2. *Example CPTs of the Bayesian Network in Figure 1(a)*

probability table $CPT(i|\mathbf{Pa}_i)$ defined on its family $F_i = \{i\} \cup \mathbf{Pa}_i$. In the following discussion, we use $T(i)$ to denote $CPT(i|\mathbf{Pa}_i)$. A *basic jointree* is similar to the classical jointree. Please refer to [3] for detailed description. In the remainder of this paper, we simply use *jointree* for *basic jointree.*

Definition 4.2. Separator: *Let $\mathcal{T} = (\mathbf{V}, \mathbf{E}', \mathbf{F})$ be a jointree. F_i is the label associated with node i. The separator associated with edge (i,j) in $\mathbf{E}'$ is defined as $separator(i,j) \stackrel{def}{=} F_{ij} \cap F_{ji}$ where F_{ij} denotes the union of all labels that are associated with nodes on the i-side of the edge (i,j).*

Definition 4.3. Message: *Messages are recursively defined as follows:*

$$message(i,j) \stackrel{def}{=} project(T(i) \prod_{k \neq j} message(k,i), separator(i,j)),$$

where $(i,j),(k,i)$ are edges in the jointree. $Message(i,j)$ encodes the information collected from node i and its neighbors except node j, and would be passed to node j along edge (i,j) during the inference. $T(i)$ is the instantiated CPT associated with node i under the given evidence [3].

Looking forward, in order to achieve the maximum performance improvement, our proposed approach employs a carefully designed order in which to make inferences, which is based on a new concept called *branch* introduced in the following context.

Definition 4.4. Branch: *Given a jointree, the query variable Q is viewed as the root of the tree. All the nodes that are connected to Q through the same neighbor N of Q can be viewed as a subtree rooted at N. We refer to this subtree as the* **branch N** *of the jointree with respect to Q. If N is a parent of Q, branch N contains all the nodes in this subtree; if N is a child of Q, branch N contains all the nodes in this subtree except N. We also define a special branch—central branch B_Q, as the collection of nodes containing Q and all its children.*

[3]The instantiated CPT of a node contains only entries that are consistent with the given evidence values. For example, given the evidence $V = False$, the instantiated CPT of node V in Table 2 contains only one entry $(False, 0.99)$.

Figure 1(b) gives a jointree of the BN in Figure 1(a). Each node in this jointree corresponds to a node in the original BN, with an annotating label containing the members of the node's family. Each edge in the jointree is annotated with its associated separator. Suppose that C is the query variable. Figure 1(b) also shows the messages passing along the paths from leaf nodes to the query variable C.

The process of inference for each tuple in Table 1 is in fact the process of (a) instantiates the leaf nodes with available evidence and (b) passing messages along edges until (c) the query node C collects messages from all its neighbors in $\mathcal{T}$. The obtained result is the joint probability distribution $Pr(C, vsxd)$. We can simply normalize this table to get the conditional probability $Pr(C = c|vsxd)$. It is then easy to conclude whether the tuple $(vsxd)$ satisfies the criterion $Pr(C = c|vsxd) > c\%$ or not.

The state of the art approach to conducting inferences on a large dataset is to successively evaluate each tuple without any reorganization of the data. This is very inefficient when the dataset is large. By making the following observations on the problem, we are able to reduce the mean processing time per tuple dramatically:

- All queries follow a common form in the sense that they share the same query variable and the same set of attributes as evidence.
- Our focus is on the conditional probability and we are only concerned with the satisfaction (or not) of the given probabilistic criterion instead of the exact probability value.
- Some attribute values may remain *unknown*.

Based on the above observations, we develop the following strategies to expedite the process.

- *Network Pruning of Conditional Probability.* The network can be further pruned to a tree that only retains relevant nodes.
- *Computation Reuse by Tuple Reordering.* The order of tuples fed into the Bayesian network can be carefully designed to eliminate redundant computation.
- *Early Termination.* Instead of calculating the exact conditional probability, we only need to determine whether its value is above some threshold. During the inference process, the range of such conditional probability can be estimated and tightened continuously to provide an opportunity to reach a conclusion before inference completes. In the worst case, we shall finish all of the computation and produce the exact conditional probability.
- The dataset can be partitioned in such a way that tuples in the same group have *unknown* values on the same set of attributes. The following algorithm is then applied to each group sequentially.

Algorithm: Inference on Large Dataset with Computation reuse and Early termination(ILDCE)

Input: A dataset $\mathcal{D}$ with attributes $\mathbf{X}$ [4], a Bayesian network $\mathcal{B}$ and a probabilistic criterion $\mathcal{C} = Pr(Q = q|\mathbf{x}) > c\%$.
Output: A subset of tuples $\mathbf{x}$ in $\mathcal{D}$ such that $\mathcal{C}(\mathbf{x}) = true$.
Preprocessing:
01. pruned network $\mathcal{B}' \leftarrow prune\_network(\mathcal{B}, Q, \mathbf{X})$;
02. jointree $\mathcal{T} \leftarrow construct\_jointree(\mathcal{B}', Q)$;
03. $\pi_b \leftarrow get\_branch\_order(\mathcal{T}, Q)$, $\pi_{\mathbf{x}} \leftarrow get\_attribute\_order(\pi_b, \mathbf{X})$;
04. $\mathcal{D}' \leftarrow cluster\_dataset(\mathcal{D}, \pi_x)$;
Batch inference:
a. **Initialize**:
05. For each branch B_K in the jointree $\mathcal{T}$; do
06. $stored(B_K) \leftarrow nil$, where $stored(B_K)$ stores the message passed from B_K to the central branch B_Q [5];
07. $valid(B_K) \leftarrow false$, where $valid(B_K)$ indicates whether the message stored at $stored(B_K)$ is valid;
08. done
09. $current\_\mathbf{x} \leftarrow nil$;
10. $num\_of\_evidence\_used \leftarrow |\mathbf{X}|$, where $|\mathbf{X}|$ is the size of $\mathbf{X}$;
11. $result \leftarrow false$.
b. **Inference**:
12. While $(next\_\mathbf{x} = get\_next\_tuple(\mathcal{D}')) \neq nil$; do
13.   If $compare\_values(next\_\mathbf{x}, current\_\mathbf{x}, num\_of\_evidence\_used) =$ true
14.   Then do
15.   If $result = true$ Then output $\mathbf{x}$;
16.   continue;
17.   done
18.   Else do
19.   For each value x in $next\_\mathbf{x}$ that is different from $current\_\mathbf{x}$; do
20.   For each branch B_K affected by x; do
21. $valid(B_K) \leftarrow false$;
22.   done
23.   done
24. $temp \leftarrow 1$;
25. $used\_evidence\_set \leftarrow \emptyset$;
26.   For $m = 1$ to $|\pi_b|$; do
27. $B_K \leftarrow \pi_b[m]$;
28. $T(B_K) \leftarrow collect\_message(B_K)$, where $T(B_K)$ is the message passed from B_K to central branch B_Q;
29. $temp \leftarrow project(temp * T(B_K), S)$, where $S = \bigcup_{k=m+1}^{k=|\pi_b|}\{$variables in branch $\pi_b[k]\} \cup Q$;
30. $used\_evidence\_set \leftarrow used\_evidence\_set \cup$ {evidence variables in B_K};
31. $n\_temp = normalize(temp)$;

[4] Since the tuples in the dataset $\mathcal{D}$ have *unknown* values on the same set of attributes, we simply make $\mathbf{X}$ refer to the set of *known* attributes.

[5] The message passed from branch B_K to the central branch B_Q is defined by Function $collect\_message(B_K)$ at the end of this algorithm.

32. If $MIN(n\_temp) > c\%$
33. Then do
34. $current\_\mathbf{x} \leftarrow next\_\mathbf{x}, result \leftarrow true, num\_of\_evidence\_used \leftarrow |used\_evidence\_set|$;
35. output $\mathbf{x}$, break;
36. done
37. If $MAX(n\_temp) < c\%$
38. Then do
39. $current\_\mathbf{x} \leftarrow next\_\mathbf{x}, result \leftarrow false, num\_of\_evidence\_used \leftarrow |used\_evidence\_set|$;
40. break;
41. done
42. done /*End of *for* loop at line 26*/
43. done /*End of *else* clause at line 18*/
44. done /*End of *while* loop at line 12*/

Function: $collect\_message(B_K)$:
45. If $valid(B_K) = true$
46. Then $result \leftarrow stored(B_K)$, return result;
47. If B_K is the central branch
48. Then do
49. $result \leftarrow$ the multiplication of instantiated $CPTs$ at Q and $Q^{'}$s children given evidence $next\_\mathbf{x}$;
50. $stored(B_K) \leftarrow result$, $valid(B_K) \leftarrow true$, return $result$;
51. done
52. If K is a parent of the query variable Q
53. Then $result \leftarrow message(K, Q)$, $stored(B_K) \leftarrow result$, $valid(B_K) \leftarrow true$, return $result$;
54. If K is a child of the query variable Q
55. Then do
56. $result \leftarrow 1$;
57. For each neighbor N of K except Q; do
58. $result \leftarrow result * message(N, K)$;
59. done
60. $stored(B_K) \leftarrow result, valid(B_K) \leftarrow true$, return $result$;
61. done

4.1 Preparing the network and dataset

This corresponds to line 01-04 in our algorithm. Because of the locality properties of a Bayesian network, not every node is relevant to all queries. Some nodes may become independent of the query variable due to the presence of evidence. Given the Bayesian network $\mathcal{B}$, the query variable Q and evidence variables $\mathbf{X}$ [6], function $prune\_network(\mathcal{B}, Q, \mathbf{X})$ prunes $\mathcal{B}$ as follows:

- Remove the edges incoming to a leaf node that is not in $\mathbf{X} \cup Q$. This is performed recursively because after the removal of some edges, interior nodes may become leaf nodes.

[6] $\mathbf{X}$ does not contain *unknown* attribute.

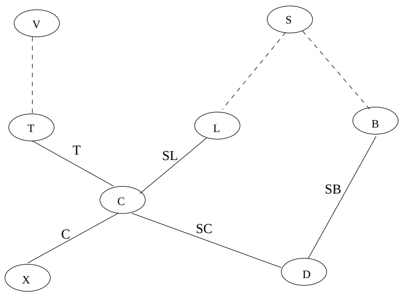


Figure 2. *Jointree built on the pruned network with query variable C and evidence V, S, X, D*

- Remove the edges outgoing from $\mathbf{X}$.

This algorithm returns the pruned network $\mathcal{B}'$ that contains only nodes reachable from Q. Since our query focuses on conditional probability, we can just consider this subgraph. The nodes that are disconnected from the query variable because of edge removal are irrelevant to the query. The jointree is thus built on the pruned graph. The edge removal step has been shown to have a great impact on reducing the size and complexity of the network for specific queries, and further saving the computation cost [3]. Figure 2 illustrates the pruned network given query variable C and evidence V, S, X, D. The dash lines indicate the removed edges. The subgraph that contains only nodes reachable from C is the network on which we construct the jointree.

In the function *construct_jointree*$(\mathcal{B}', Q)$, we adopt a similar strategy to [3] to build the jointree. Basically, this strategy is just to remove enough edges from the network so that it become singly-connected. There are two advantages to building a jointree in such a way. First, it's simple and easy to understand; secondly, it keeps the topology of the original graph, since all the edges in the jointree are edges in the original graph. This structure is useful for detecting early termination.

While constructing an optimal jointree is not always feasible, it may be possible to construct a jointree with relatively good quality. There are several concerns in constructing a good jointree. First of all, it is desirable to construct a jointree such that the number of variables shared by different branches is small. The intuition behind this is that the variables shared by two branches are retained in the separators along the path between these branches, and the size of the separators determines the size of messages and further determines the complexity of the jointree algorithm. By constructing a jointree with a small number of shared variables, we can achieve a relatively low computation complexity. Secondly, it is better to construct a relatively "balanced" jointree so that no branch contains a preponderance number of evidence variables. The change of evidence values in a branch causes changes in the corresponding messages and a large number of evidence variables implies more computation. By constructing a "balanced" jointree, we can reduce the expected recomputation cost incurred by evidence change.

Compared with the state of art approach, which does not address how the dataset is organized, our algorithm rearranges the data set in a specific order. We shall see in a later section that the way the tuples are stored in the dataset and the order in which the evidence values are fed in the inference process affect the performance significantly. The improvement in performance can well compensate for the cost of dataset preprocessing.

In our algorithm, the function $cluster\_dataset(\mathcal{D}, \pi_x)$ reorders the dataset $\mathcal{D}$ into $\mathcal{D}^{'}$ according to the order of attributes π_x. In $\mathcal{D}^{'}$, the tuples with the same values on the first element in π_x are grouped together. Among each such group, the tuples with the same values on the second element in π_x are grouped together, and so on. The attribute values in a tuple are rearranged according to their order of appearance in π_x. In fact, we do not need to physically change the order of values, but simply define a map between the physical order of these values and π_x. The order π_x can be decided by π_b—— the order of branches to be computed in the inference process. In π_b, the central branch is always chosen as the first branch to be computed. The order of other branches is chosen to maximize the savings of computation reuse or early termination.

4.2 Making inference on large dataset

The inference process is to collect the messages along the paths from the leaf nodes of each branch to the central branch. During this process, we apply two techniques: computation reuse by tuple reordering and early termination.

The computation reuse is relatively straightforward. In our algorithm, the message passed from each branch B_K to the central branch B_Q is stored at $stored(B_K)$, with an additional variable $valid(B_K)$ indicating whether the stored message is valid or not. Since we are making inference for a series of queries with the same criterion, the query variable and evidence variables remain unchanged, and so do the structure of the jointree and the separators associated with edges in the jointree $\mathcal{T}$. According to the definition of *message*, if no evidence value in branch B_K changes, the message passed from this branch to the central branch will not change. By storing this message, we can directly use it during the inference of successive tuples if the evidence on this branch remain the same. In our algorithm, line 05-08 initialize the validity of the stored message for each branch to false. Function $collect\_message(B_K)$ (starting at line 45) outlines the computation reuse in computing the message passed from a branch B_K to the central branch.

The idea of early termination is to compute a bound for the final conditional probability using the intermediate result. This is performed by line 24-42 in our algorithm. We first compute the central branch B_Q and set the intermediate result table *temp* as the message received at B_Q. For each of the remaining branches B_K, we get a message $T(B_K)$, which is the message passed from branch B_K to the central branch, and multiply it by *temp*. It is guaranteed that no matter what further information will be included, the final probability will not exceed the bound defined by the minimal and maximal conditional probability in n_temp, the normalized version of *temp* (at line 31). Once this range is completely below the threshold, we can conclude that the current tuple does not satisfy the criterion;

similarly, we can conclude that the current tuple satisfies the criterion if the range is completely above the threshold. Moreover, this range continues to be refined as messages from more branches are computed and multiplied to *temp*. After messages from all the branches are integrated, we obtain the exact conditional probability.

Because of the order we arrange the dataset, if early termination happens (that is, we know *result* is either *true* or *false* by using only the first *num_of_evidence_used* attribute values), we may tell the results of the successive tuples for free if they have the same prefix as the current tuple. Function *compare*(*next_***x**, *current_***x**, *num_of_evidence_used*) (at line 13) tests whether *next_***x** and *current_***x** have the same values on the first *num_of_evidence_used* attributes. If this function returns *true*, we can directly copy the result of *current_***x** to *next_***x**.

The general process of inference for the example in Figure 1(a) proceeds as follows. The criterion is $Pr(C = c|VSXD) > c\%$. The inference is made on the jointree in Figure 2. This jointree can be viewed as rooted on the query variable C with four branches: branch B_T, B_L, B_D and the central branch B_C.

(a) Multiplying the CPTs in the central branch B_C, we get:

$$temp(TLCxBd) = CPT(C|TL)CPT(x|C)CPT(d|CB).$$

$temp(TLCxBd)$ can be converted to a conditional probability table on C:

$$temp(TLCxBd) \Rightarrow n\_temp(C|TLxBd).$$

It's easy to see that

$$MIN_{TLxBd} n\_temp(C|TLxBd) \leq P(C|vsxd) \leq MAX_{TLxBd} n\_temp(C|TLxBd).$$

The final conditional probability can be bounded using the minimal and maximal conditional probabilities of $n\_temp(C|TLxBd)$. For query $P(c|vsxd) > c\%$, if $MIN_{TLxBd} n\_temp(c|TLxBd) > c\%$, we know that the current tuple($vsxd$) satisfies the query without further computation. Similarly, if $MAX_{TLxBd} n\_temp(c|TLxBd) < c\%$, we can tell in advance that the current tuple ($vsxd$) will not satisfy the query. If we cannot determine the satisfaction of the criterion on the current tuple, we have to calculate the messages passed from other branches.

(b) Calculate the message passed from node T to C:

$$T(B_T) = message(T, C) = project(CPT(T|v), T).$$

Multiplying $T(B_T)$ with $temp(TLCxBd)$ and projecting the result on attributes $VLCXBD$, we obtain

$$temp(vLCxBd) = project(temp * T(B_T), VLCXBD).$$

Similarly, we have

$$temp(vLCxBd) \Rightarrow n\_temp(C|vLxBd).$$

By the same reasoning, we can prove that

$$\begin{aligned} MIN_{TLxBd}n\_temp(C|TLxBd) \quad &\leq \quad MIN_{vLxBd}n\_temp(C|vLxBd) \\ &\leq P(C|vsxd) \leq \\ MAX_{vLxBd}n\_temp(C|vLxBd) \quad &\leq \quad MAX_{TLxBd}n\_temp(C|TLxBd). \end{aligned}$$

With the more strict bound

$$[MIN_{vLxBd}n\_temp(C|vLxBd), MAX_{vLxBd}n\_temp(C|vLxBd)],$$

we may be able to terminate the inference process for the current tuple. Otherwise, we need to choose another branch to calculate.

(c) Choose branch $L \to C$ to calculate, multiply $T(B_L)$ with $temp(vLCxBd)$ and project the result on $VSCXBD$, we obtain

$$temp(vsCxBd) \Rightarrow n\_temp(C|vsxBd).$$

The bound $[MIN_{vsxBd}n\_temp(C|vsxBd), MAX_{vsxBd}n\_temp(C|vsxBd)]$ is contained in

$$[MIN_{vLxBd}n\_temp(C|vLxBd), MAX_{vLxBd}n\_temp(C|vLxBd)].$$

(d) Choose the final branch $B \to D$, calculate $T(B_D)$, multiply it with $temp(vsCxBd)$ and project on $VSCXD$, we have

$$temp(vsCxd) \Rightarrow n\_temp(C|vsxd),$$

which is the exact conditional probability we need.

From the above procedure, we can see that the minimal conditional probability of the intermediate result is monotonically increasing as we calculate from one branch to another, while the maximal conditional probability is monotonically decreasing. The continuously refined range provides us the opportunity to terminate the inference as soon as the range does not overlap the threshold value.

Let's revisit the dataset in Table 1. If there are rearranged according to the order $\pi_x = (X, D, V, S)$ into Table 3. We follow the steps (a) (b) (c) (d) to process the inference. The inference process for Patient 1 terminated at step (c), where we get a bound $[0.000549281, 0.00340798]$ for the final conditional probability, which is below the threshold 40%. Patient 2 has the same values on attributes XDV, so that it reused the intermediate result of Patient 1 for the first two steps. At step (c), it has a bound $[0.00330038, 0.0201878]$, which causes the termination of the current inference process, since this bound is completely below 40%. Patient 0 did not reuse the previous result because of the change on evidence values. However, it got an early termination at step (b), where the bound $[0.570136, 1]$ is completely above the threshold. Up to step (b), we made use of the values on evidence (XDV) only. Because of the early termination, the value of S will not change the result.

This gives us the benefit of obtaining the result for Patient 3 directly without any computation, since Patient 3 and Patient 0 agree on the values for (XDV).

Algorithm ILDCE uses both techniques: computation reuse by tuple reordering and early termination. It needs to multiply the CPTs at the query variable Q and Q's children first. This may not always be ideal as we shall see in the experimental results. A variant of this algorithm is to use computation reuse only. Besides removing the code related to early termination in algorithm ILDCE, we need to modify the function $collect\_message(B_K)$ slightly as follows.

Patient	Visit To Asia	X-Ray Result	Smoking	Dyspnea
Patient 1	False	Normal	False	Present
Patient 2	False	Normal	True	Present
Patient 0	True	Abnormal	False	Present
Patient 3	True	Abnormal	True	Present
...	...	...	...	...

Table 3. *tuple reorder*

Function: $collect\_message1(B_K)$:

If $valid(B_K) = true$

Then $result \leftarrow stored(B_K)$, return result;

If B_K is the central branch

Then $result \leftarrow$ the instantiated CPT at Q given evidence $next\_\mathbf{x}$, $stored(B_K) \leftarrow result$, $valid(B_K) \leftarrow true$, return $result$;

Else $result \leftarrow message(K, Q)$, $stored(B_K) \leftarrow result$, $valid(B_K) \leftarrow true$, return $result$;

5 Experiment results

The experiments reported in this section contain ten randomly generated Bayesian networks. Each of them contains 30 nodes and 60 edges. For each BN, we arbitrarily chose a query variable that has multiple children and parents and 10 random evidence variables. The experiments were run on Sun Ultra 30, with 128M memory.

Table 4 lists the cost of inference by the state of art approach(SA), using computation reuse only(CR), and the approach using early termination plus computation reuse(ET+CR). All of these approaches (including SA) are based on the pruned network. The cost of SA is the execution time. ET+CR contains 5 columns, each of which corresponds to a conditional probability threshold.

The experiments show that, in most cases, CR and ET+CR have great performance improvement compared to SA. In ET+CR, when the threshold changes from 0.1 to 0.9 for each case, the cost curve usually rises up and then goes down. This is because the inferences for most tuples can be terminated in early stages when the threshold $c\%$ is easily beyond the possible probability range of the query variable.

The above experiments also show that ET+CR does not always outperform

Case#	*#instance*	*SA**	*CR*	*ET+CR*				
				0.1	0.3	0.5	0.7	0.9
Case 1	6912	449.28	418	21	42	218	49	2
Case 2	27648	200780	80154	6817	7350	7348	7618	7361
Case 3	13824	5819.9	366	966	1608	1670	1552	580
Case 4	15552	179719	117439	4482	6644	6049	4274	538

Table 4. *Result table*

CR (e.g., Case 3). The reason is that, for CR, given a jointree, we multiply the messages passing from each branch to the query variable Q. However, in ET+CR, to make use of early termination, we first calculate an intermediate table $T(B_Q)$ for the central branch by multiplying the CPT at the query variable Q with the CPTs at Q's children. We can then derive a bound for the conditional probability of Q from $T(B_Q)$. Compared with CR, the messages passing from other branches in ET+CR are multiplied by this intermediate table instead of the CPT at Q. $T(B_Q)$ could be a much larger table than the CPT at Q when Q has many children. In this situation, the performance of ET+CR is impacted. Even if we save the cost of inference for some tuples due to early termination, the total execution time of ET+CR can still be larger than that of CR. Nevertheless, if the size of $T(B_Q)$ at the central branch is comparable to that of the CPT at Q (i.e, Q has many parents but few children), and the early termination does happen, the saving of ET+CR on computation cost is also significant.

We also apply our techniques to a larger Bayesian network, which consists of 100 nodes and 300 edges. We found that the relative benefits of the early termination and computation resuse technique remains similar to that of the smaller network. Due to space limitations, we omit the detailed performance analysis of the larger network.

6 Conclusion

In this paper, we investigate a new data mining problem and introduce a paradigm that supports user interference in the mining process. We employ Bayesian network to encode uncertain information and propose three techniques——network pruning on conditional probability, computation reuse by tuple reordering and early termination—— to reduce the average cost per inference. Empirical tests show that our proposed techniques can save as much as 95% computation time. Last but not least, our algorithm is applicable to both online and offline applications. With minor modification, the algorithm can support an interactive mining environment that allows user to better control the course of inference.

Acknowledgment

The work of Yi Xia and Richard Muntz was partially supported by NSF grants IIS-0086116, ANI-0085773, EAR-9817773, and an IBM Faculty Partnership Award.

Bibliography

[1] B. Abramson. Arco1: an application of belief networks to the oil market. In *7th Conference on Uncertainty in Artificial Intelligence*, pages 1–8, 1991.

[2] B. Abramson, J. Brown, A. Murphy, and R. L Winkler. Hailfinder: A bayesian system for forecasting severe weather. *International Journal of Forecasting*, pages 12, 57–71, 1996.

[3] Adnan Darwiche. Dynamic jointrees. In *14th Conference on Uncertainty in Artificial Intelligence*, pages 97–104, 1998.

[4] K. Dechter. Bucket elimination: a unifying framework for probabilistic inference. In *Uncertainty in Artificial Intelligence*, pages 211–219, 1996.

[5] K. Dechter. Mini-buckets: A general scheme for approximating inference. ICS Technical Report, 1998.

[6] D. Heckerman, E. Horvitz, and B. Nathwani. Towards normative experts systems: Part i. the pathfinder project. *Methods of Information in Medicine*, pages 31, 90–105, 1992.

[7] D. Heckerman and B. Nathwani. Towards normative experts systems: Part ii. probability-based representations for efficient knowledge acquisition and inference. *Methods of Information in Medicine*, pages 31, 106–116, 1992.

[8] Cecil Huang and Adnan Darwiche. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, pages 225–263, 1996.

[9] F. V. Jensen, S.L. Lauritzen, and K. G. Olesen. Bayesian updating in recursive graphical models by local computation. *Computational Statistics Quarterly*, pages 269–282, 4, 1990.

[10] Judea Pearl. Evidential reasoning using stochastic simulation. *Artificial Intelligence*, pages 245–257, 32, 1987.

[11] Judea Pearl. Probabilistic reasoning in intelligent systems: Networks of plausible inference. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1988.