An Incremental Algorithm for Tree-shaped Bayesian Network Learning

Josep Roure Alcobé [1]

Abstract. Incremental learning is a very important approach to learning when data is presented in short chunks of instances. In such situations, there is an obvious need for improving the performance and accuracy of knowledge representations or data models as new data is available. It would be too costly, in computing time and memory space, to use the batch algorithms processing again the old data together with the new one.

We present in this paper an incremental algorithm for learning tree-shaped Bayesian Networks. We propose a heuristic able to trigger the updating process when data invalidates, in some sense, the current structure. The algorithm rebuilds the network structure from the branch which it is found to be invalidated. We will experimentally demonstrate that the heuristic is able to obtain almost optimal tree-shaped Bayesian Networks while saving computing time.

1 Introduction

During the nineties, there have been a great deal of effort in developing algorithms for learning Bayesian Networks from data. See, Buntine [3] for an overview of the literature and Heckerman [7] for a tutorial. Most of this work has focused in non-incremental learning algorithms. Such algorithms, also called batch, infer a Bayesian Network based on the entire set of available data.

In this paper we focus on incremental learning of Bayesian Networks. This sort of learning attempts to update current Bayesian Networks in response to newly observed data instances. Langley [9] stated that an algorithm is incremental if (1) it inputs one training experience at a time, (2) does not reprocess any previous data instances, and (3) retains only one knowledge structure in memory.

Each of these three constraints aims at clear objectives. The first wants incremental algorithms to be able to output a Bayesian Network at any moment of the learning process. The second keeps low and constant the time required to process each data instance over all the data set. And finally, the third constraint wants learning algorithms not to do unreasonable memory demands.

We propose, in this paper, an incremental heuristic. We could process together the old and new data with a batch algorithm in order to learn a new Bayesian Network. That would require lot of computing time and to store all the data instances. The main objective of our approach is to spend less time than a batch approach and to store only that information, from the data set, useful for learning.

We focus on tree-shaped Bayesian Networks because they are simple structures easier to learn than general ones and because the amount of information needed from the data set can be stored in main memory as we will argue later on. We apply the heuristic to the algorithm proposed by Chow and Liu [4] which learns tree structures.

The Chow and Liu's algorithm, CL algorithm from now on, builds a maximal cost tree introducing branches into the tree in decreasing cost order. Roughly speaking, our heuristic states that new data invalidates the tree-shaped structure when the branches are not anymore in decreasing cost order. Then, the tree is rebuilt from the first branch found to be in a bad position into the order. In this way, our algorithm, can both detect the need of updating and update the current network. We will call our proposal ACO heuristic (Arches in Correct Order). Note that we use, in the name, arches instead of branches for a reason that will be clear later on.

We relax, in our work, the first Langley's constraint allowing to process a short chunk of data instances instead of each single instance. This point does not break the spirit of the constraint as one single data instance slightly modifies the information available, and it would almost surely not trigger the updating process.

The rest of this paper is organized as follows. In Section 2, we introduce the batch algorithm proposed by Chow and Liu [4]. In Section 3, we present our incremental algorithm and the ACO heuristic to guess when a network should be updated. In Section 4, we empirically evaluate our proposal. Finally, in Section 5, we conclude with some remarks and with a short discussions of related work.

2 Chow and Liu's batch algorithm

In this section we study the algorithm proposed by Chow and Liu [4]. They designed an algorithm to estimate the underlying n-dimensional discrete probability distribution from a set of samples. The algorithm yields as an estimation the product of $n-1$ second order distributions that optimally approximates the probability distribution. This product can also be formulated like a distribution of $n-1$ first order dependence relationships among the n variables, forming a tree dependence structure.

The algorithm uses the mutual information as closeness measure between $P(\mathbf{X})$ and $P_\tau(\mathbf{X})$, where $P(\mathbf{X})$ is the probability distribution from a set of samples, and $P_\tau(\mathbf{X})$ is the tree dependence distribution. It is an optimization algorithm that gives the tree distribution closest to the distribution from the samples. Let us give some notation in order to explain the Chow and Liu's measure and algorithm.

Let $(m_1, \cdots, m_n)$ be a permutation of integers $1, 2, \cdots, n$, let $j(i)$ be a mapping with $0 \leq j(i) \leq i$, let $\mathcal{T} = (\mathbf{X}, E)$ be a dependence tree where $\mathbf{X}(\mathcal{T}) = \{X_{m_i} | 1, 2, \cdots, n\}$ is the set of nodes, $E(\mathcal{T}) = \{(X_{m_i}, X_{m_{j(i)}}) | 1, 2, \cdots, n\}$ is the set of branches, and where X_0 is the *null* node. If we now assign the mutual information between two variables, $I(X_{m_i}; X_{m_{j(i)}})$, as a cost to every dependence tree branch, the maximum-cost dependence tree is defined

[1] Departament d'Informàtica i Gestió, Escola Universitària Politècnica de Mataró, Avda. Puig i Cadafalch 101-111, 08303 Mataró, Catalonia, Spain, roure@eupmt.es

as the tree $\mathcal{T}$ such that for all $\mathcal{T}'$ in $\mathcal{T}_n$, $\sum_{i=1}^{n} I(X_{m_i}; X_{m_{j(i)}}) \geq \sum_{i=1}^{n} I(X_{m_i}; X_{m_{j'(i)}})$. Where $\mathcal{T}_n$ stands for the set of trees with n variables.

Chow and Liu applied some transformations to Kullback-Leibler divergence $D_{KL}(P, P_\tau)$, and obtained the following expression:

$$D_{KL}(P, P_\tau) = -\sum_{i=1}^{n} I(X_{m_i}; X_{m_{j(i)}}) + \sum_{i=1}^{n} H(X_{m_i}) - H(\mathbf{X})$$

From the expression above, it is observed that $H(X_{m_i})$ and $H(\mathbf{X})$ for all i are independent of the tree dependence distribution. Thus, since $D_{KL}(P, P_\tau)$ is non-negative, minimizing the closeness measure $D_{KL}(P, P_\tau)$ is equivalent to maximizing the term $\sum_{i=1}^{n} I(X_{m_i}; X_{m_{j(i)}})$. This result allowed Chow and Liu to use the Kruskal algorithm for the construction of trees of maximum total cost where $I(X_{m_i}; X_{m_{j(i)}})$ may represent the distance cost from node X_{m_i} to node $X_{m_{j(i)}}$. An undirected graph is formed by starting with a graph without branches and adding a branch between two nodes with the highest mutual information. Next, a branch is added which has maximal mutual information associated and does not introduce a cycle in the graph. This process is repeated until the $(n-1)$ branches with maximum mutual information associated are added as seen in Algorithm 1.

In this paper, we give a direction to all the branches of the tree. We take as the root of the tree one of the nodes of the first branch and the direction of the branches introduced afterwards goes from the node already into the structure to the one recently introduced. As all branches have direction we will call them arches and will be represented as ordered pairs of nodes $(X_{m_i}, X_{m_{j(i)}})$.

Algorithm 1 CL

Require: a database D on $\mathbf{X} = \{X_{m_1}, \cdots, X_{m_n}\}$ variables
Ensure: $\mathcal{T}$ be a dependence tree structure
  Calculate $SUFF_D(\mathcal{T})$
  $\mathcal{T} = (\mathbf{V}, E)$ the empty tree where $\mathbf{V}(\mathcal{T}) = \{\emptyset\}$ and $E(\mathcal{T}) = \{\emptyset\}$
  Calculate costs for every pair $I(X_{m_i}; X_{m_j})$
  Select the maximum cost pair (X_{m_i}, X_{m_j})
  $\mathbf{V}(\mathcal{T}) = \{X_{m_i}, X_{m_j}\}$; $E(\mathcal{T}) = \{(X_{m_i}, X_{m_j})\}$
  repeat
    $B(\mathcal{T}) = \{(X_{m_i}, X_{m_j}) \mid ((X_{m_i}, X_{m_k}) \in E(\mathcal{T}) \vee (X_{m_k}, X_{m_i}) \in E(\mathcal{T})) \wedge X_{m_j} \notin \mathbf{V}(\mathcal{T})\}$
    Select the max cost pair (X_{m_i}, X_{m_j}) from $B(\mathcal{T})$
    $\mathbf{V}(\mathcal{T}) = \mathbf{V}(\mathcal{T}) \cup \{X_{m_j}\}$
    $E(\mathcal{T}) = E(\mathcal{T}) \cup \{(X_{m_i}, X_{m_j})\}$
  until $(\mathbf{V} = \mathbf{X})$

3 An incremental algorithm

In this section we propose an incremental algorithm in order to learn a tree-shaped Bayesian network when new training data instances are available. Our algorithm revises an already learnt tree-shaped Bayesian Network without processing the old data instances. It actually modifies the old structure only when it is invalidated, in some sense, by the new data. The main objective is to spend less computing time than a batch approach that learnt a structure from scratch each time new data was available.

We introduce some notation before the explanation of our algorithm. Let $N_X^D(x)$ be the number of instances in D where $X = x$. Let $\widehat{N}_X^D$ be the vector of numbers $N_X^D(x)$ for all values of X. We call the vector $\widehat{N}_X^D$ the *sufficient statistics* of the variable X, $suff_D(X)$. In the same way, the *sufficient statistics* of the tree $\mathcal{T}$, $suff_D(\mathcal{T})$, are defined as the set of vectors $\widehat{N}_{X_{m_i}, X_{m_{j(i)}}} \forall i : 0 \leq i \leq n$.

To find the maximal cost tree we need the vector numbers $\widehat{N}^D_{X_{m_i}, X_{m_{j(i)}}}$ for all the pairs of variables in $\mathbf{X}(\mathcal{T})$, we will call this set of numbers $SUFF_D(\mathcal{T})$. Note that $SUFF_{D \cup D'}(\mathcal{T})$ can be calculated as $\widehat{N}^D_{\mathbf{X}} \oplus \widehat{N}^{D'}_{\mathbf{X}}$, where $\oplus$ stands for the addition of vector components.

We divide our algorithm into two steps. In the first step, the algorithm calculates the *sufficient statistics* for both old D and new D' data instances, and in the second, it revises the tree structure according to the new *sufficient statistics*.

In the first step of the algorithm, we assume that *sufficient statistics* of the old data set D are stored. Thus, in order to recover the *sufficient statistics*, $SUFF_{D \cup D'}(\mathcal{T})$, of the whole set of data instances the algorithm does not need to go through the old ones.

The second step of the algorithm uses a heuristic in order to decide whether to update the tree structure. The heuristic decides to update the structure when the arches are not in correct order. When the tree is built for the very first time using the CL algorithm, arches are introduced into the tree structure in decreasing cost order. This order $\mathcal{O}$ is stored. When new data instances D' are presented, the cost $I(X_{m_i}; X_{m_{j(i)}})$ for each branch is calculated again using the new *sufficient statistics* $SUFF_{D \cup D'}(\mathcal{T})$, and only when the order $\mathcal{O}$ does not hold anymore, that is, when arches are not in decreasing cost order, the structure is updated.

Algorithm 2 ACO heuristic

Require: a database D' on $\mathbf{X} = \{X_{m_1}, \cdots, X_{m_n}\}$ variables a tree structure $\mathcal{T}$, an order $\mathcal{O}$ of branches and $SUFF_D(\mathcal{T})$
Ensure: $\mathcal{T}'$ be a dependence tree structure
  Calculate $SUFF_{D \cup D'}(\mathcal{T})$
  $\mathcal{T}' = (\mathbf{V}, E)$ the empty tree where $\mathbf{V}(\mathcal{T}') = E(\mathcal{T}') = \{\emptyset\}$
  Let X_{m_h} be the root of $\mathcal{T}$
  $B(\mathcal{T}) = \{(X_{m_h}, X_{m_j}) \mid (X_{m_h}, X_{m_j}) \in E(\mathcal{T})\}$
  continue=false; *k=0*
  if $((X_{m_i}, X_{m_j})_{\mathcal{O}(1)} = = arg\ max_{(X_{m_r}, X_{m_s}) \in B(\mathcal{T}) \cap E(\mathcal{T})} I(X_{m_r}, X_{m_s}))$ **then**
    $\mathbf{V}(\mathcal{T}') = \{X_{m_h}, X_{m_j}\}$; $E(\mathcal{T}') = \{(X_{m_h}, X_{m_j})\}$
    continue=true; *k=2* be the number of branches added (+ 1)
  end if
  while (*continue*) and $(k \leq |E(\mathcal{T})|)$ **do**
    $B(\mathcal{T}_{\mathcal{O}(k)}) = \{(X_{m_i}, X_{m_j}) \mid ((X_{m_i}, X_{m_k}) \in E(\mathcal{T}_{\mathcal{O}(k)}) \vee (X_{m_k}, X_{m_i}) \in E(\mathcal{T}_{\mathcal{O}(k)})) \wedge X_{m_j} \notin \mathbf{V}(\mathcal{T}_{\mathcal{O}(k)})\}$
    if $((X_{m_i}, X_{m_j})_{\mathcal{O}(k)} = = arg\ max_{(X_{m_r}, X_{m_s}) \in B(\mathcal{T}_{\mathcal{O}(k)}) \cap E(\mathcal{T})} I(X_{m_r}, X_{m_s}))$ **then**
      $\mathbf{V}(\mathcal{T}') = \mathbf{V}(\mathcal{T}') \cup \{X_{m_j}\}$
      $E(\mathcal{T}') = E(\mathcal{T}') \cup \{(X_{m_i}, X_{m_j})\}$; *k++*
    else
      continue=false
    end if
  end while
  if $(k \leq |V(\mathbf{X})|)$ **then**
    Continue building $\mathcal{T}'$ using the original CL algorithm
  end if

More precisely our algorithm, see Algorithm 2, inspects the arches in the order $\mathcal{O}$ they were added into the tree. When an arch

$(X_{m_i}, X_{m_{j(i)}})_{\mathcal{O}(k)}$ at the k-th position in $\mathcal{O}$ has not the highest cost among all candidate arches present into the former structure, the tree is rebuilt from that arch using the original CL algorithm.Formally, when the arch at the k-th position $(X_{m_i}, X_{m_{j(i)}})_{\mathcal{O}(k)} \neq$ *arg max*$_{(X_{m_k}, X_{m_l}) \in B(\mathcal{T}_{\mathcal{O}(k)}) \cap E(\mathcal{T})} I(X_{m_k}, X_{m_l})$. Where $\mathcal{T}_{\mathcal{O}(k)}$ stands for the tree built only with the first $k-1$ arches of the order $\mathcal{O}$ and $B(\mathcal{T}_{\mathcal{O}(k)})$ stands for the set of arches that do not introduce any cycle in $\mathcal{T}_{\mathcal{O}(k)}$, $B(\mathcal{T}_{\mathcal{O}(k)}) = \{(X_{m_i}, X_{m_j}) \mid ((X_{m_i}, X_{m_k}) \in E(\mathcal{T}_{\mathcal{O}(k)}) \vee (X_{m_k}, X_{m_i}) \in E(\mathcal{T}_{\mathcal{O}(k)})) \wedge X_j \notin \mathbf{V}(\mathcal{T}_{\mathcal{O}(k)})\}$.

Note, it may happen that (X_{m_i}, X_{m_k}) has the maximum cost among the arches in $B(\mathcal{T}_{\mathcal{O}(k)}) \cap E(\mathcal{T})$ and not among the ones in $B(\mathcal{T}_{\mathcal{O}(k)})$. In such situation, the ACO heuristic and the CL algorithm would not recover the same tree structure.

We assume, in this paper, that we can store in main memory all the *sufficient statistics SUFF*$(\mathcal{T})$. If all variables in $\mathbf{X}$ have the same number of values $val(X)$ then the $|$*suff*$(X_{m_i}, X_{m_{j(i)}})| = |\widehat{N}_{X^D_{m_i}, X_{m_{j(i)}}}| = val(X)^2$, and the number of values of the *SUFF*$(\mathcal{T})$ of all alternative trees in $\mathcal{T}_n$ is $\binom{n}{2} \times val(X)^2$ which is quadratic in the number of variables and the number of values. In our experiments, we used sparse ADtrees [10], with only two levels, as a compact representation of the *sufficient statistics*.

If we had so many variables in the data so that the storage became infeasible, we would not be able to use this incremental algorithm. Even though, in such situation we could use some method to reduce the number of attributes excluding the less relevant ones, see for example [12].

4 Experiments

We conducted several experiments in order to compare the repeated use of the batch CL algorithm against our incremental ACO heuristic. We presented data instances to both algorithms in chunks of 100 and compared the Bayesian Networks produced during all the learning process. We used five well-known datasets, one generated from the alarm network [1] (37 variables and 20.000 instances), and four from the UCI machine learning repository [11]: Nursery (9 variables and 12.960 instances), Chess (7 variables and 28.056 instances), Mushroom (23 variables and 8.124) and DNA (61 variables and 3.190 instances).

We presented the instances to the algorithms in three different kind of orders. Namely, an order where similar instances are consecutive, another where dissimilar instances are consecutive, and finally a random order. We used five different orders of each kind to run both algorithms, and all numbers presented in the tables of this section are the mean and the standard deviation of the quantity being analyzed.

We used these three different kind of orders because it is widely reported in the literature that incremental algorithms may yield different results when the same instances are presented in different orders. It is seen, in the field of incremental clustering, that when dissimilar instances are consecutively presented, results are much better than when similar instances are presented together [5]. This may occur because, in the former case, initial observations are sampled from different parts of the description space leading initial structures to approximate the actual probability distribution of the dataset, while in the later, rather skewed structures may be built at the beginning, biasing the rest of the learning process.

4.1 Computational time gain

The main objective of the algorithm proposed in this work was to reduce the time spent in learning a new network structure when the system already learned one from past data. In Table 1, we show the CPU clock ticks spent by both algorithms in order to learn all the network structures produced when data sets are presented in chunks of 100 instances.

We can see that the batch algorithm uses much more CPU clock ticks than the ACO heuristic. We also note from the table that the more attributes data sets have the greater the gain is. Compare Alarm against Nursery and Chess results. And also, we can see that the gain grows with the number of data instances (see Alarm results). This last point is due to the fact than when many data instances have already been processed, the new data instances slightly modify the probability distribution of the database and therefore the network structure does not need to be updated.

Another cause which may influence the time spent is the order in which the instances are presented. Usually, when dissimilar instances are presented consecutively, the network structures learned from data are not good models of the probability distribution of the entire database. Thereof, the incremental algorithm spends more time as it must update the network structure. We see, at Table 1, that the time spent is usually lower when random orders are used.

Table 1. CPU clock ticks spent in learning

		Batch	Incremental	Gain (%)
Alarm	Rand	1296 (24.22)	250 (14.69)	80.71
	Sim	1382 (24.23)	496 (16.23)	64.11
	Diss	1468 (23.29)	306 (14.79)	79.16
Nursery	Rand	56 (7.32)	34 (10.27)	39.28
	Sim	56 (10.92)	44 (7.40)	21.42
	Diss	44 (9.87)	40 (9.15)	9.09
Chess	Rand	316 (17.01)	170 (15.10)	46.20
	Sim	344 (12.05)	212 (15.46)	38.37
	Diss	336 (18.07)	154 (16.22)	54.16
Mushroom	Rand	440 (7.52)	224 (12.59)	49.09
	Sim	400 (14.72)	278 (15.06)	30.50
	Diss	400 (17.69)	284 (14.54)	29.00
DNA	Rand	1506 (17.17)	1418 (10.89)	3.18
	Sim	1520 (16.05)	1452 (13.04)	4.47
	Diss	1594 (12.19)	1458 (13.76)	8.53

4.2 Quality of the recovered structures

In Figure 1, we show the behavior of our heuristic along the learning process where the algorithm is fed with chunks of 100 data instances, and using a random data order. We compare the structures obtained with our incremental proposal against the ones obtained with the CL batch algorithm. In the figure, there is a graphic of the data sets, Alarm, Mushroom and DNA. We do not show graphics of Nursery and Chess because of the lack of space.

Each graphic presents three curves. The first curve, shows the first arch which is not in decreasing cost order. When the number shown coincides with the number of attributes, it means that all arches are in decreasing cost order and consequently the structure is not updated. This curve gives an idea of when our heuristic detects that the structure must be updated. The second, shows the number of different arches between the structures learnt by the batch algorithm and

the ones learnt by our incremental proposal. This curve gives us an idea of how well our incremental approach approximates the best solution. Finally, the third curve, shows the number of arches that are different between the former and the current tree structure learnt with the batch algorithm. This curve gives an idea of the degree in which the new 100 data instances make the current structure to change.

Looking at the figure, we discover that our incremental algorithm approximates very well the best solution. It is able to detect when the structure should be updated and is updated correctly even when the arch found to be incorrectly ordered is one of the last ones in the order.

The third curve shows that, at the early stages of the learning process, when few data instances have already been processed, the structure changes quite a lot and that the number of changed arches tend to decrease as more data is processed. This is very well seen at the graphic of the DNA dataset, which has few data instances. Even in this case the incremental algorithm learns structures very close to the best one. If we look back to Table 1, we can see that the incremental algorithm saves, in this case, little time as it must trigger the CL algorithm very often at firsts arches, building almost the entire tree.

We also want to note that the quality, $\sum_{i=1}^{n} I(X_{m_i}; X_{m_{j(i)}})$, of the final network structures obtained with the batch and the incremental algorithm is almost the same. See Table 2. This is easily explained as the final structures are almost the same as shown in Figure 1. In the experiments, the last *n mod 100* data instances in the order were not processed.

Table 2. Quality, $\sum_{i=1}^{n} I(X_{m_i}; X_{m_{j(i)}})$, of final network structures

		Batch	Incremental
Alarm	Rand	7.248(0.009)	7.248(0.008)
	Sim	7.266(0.001)	7.266(0.001)
	Diss	7.232(0.003)	7.232(0.003)
Nursery	Rand	0.894(0.004)	0.896(0.000)
	Sim	0.896(0.002)	0.896(0.002)
	Diss	0.896(0.005)	0.896(0.005)
Chess	Rand	0.775(0.051)	0.786(0.000)
	Sim	0.786(0.000)	0.786(0.000)
	Diss	0.787(0.004)	0.787(0.004)
Mushroom	Rand	7.910(0.026)	7.915(0.003)
	Sim	7.921(0.000)	7.921(0.000)
	Diss	7.887(0.028)	7.887(0.028)
DNA	Rand	3.571(0.015)	3.571(0.015)
	Diss	3.527(0.010)	3.527(0.010)
	Sim	3.505(0.011)	3.505(0.011)

4.3 Learning curves

In this section we study the leaning curve of the incremental algorithm when data is presented in different orders. In Figure 2, we can see the learning cost of the tree structures for the Mushroom data set presented in three different orders. In the first order data instances are presented randomly, in the second similar instances are presented consecutively and, in the third dissimilar instances are presented consecutively.

The graphic (a) shows the quality of the trees calculated with the data explored so far, while the graphic (b) shows the quality of the tree calculated using the whole data set. This last graphic shows how well the discovered Bayesian Networks approximate the distribution of the whole dataset.

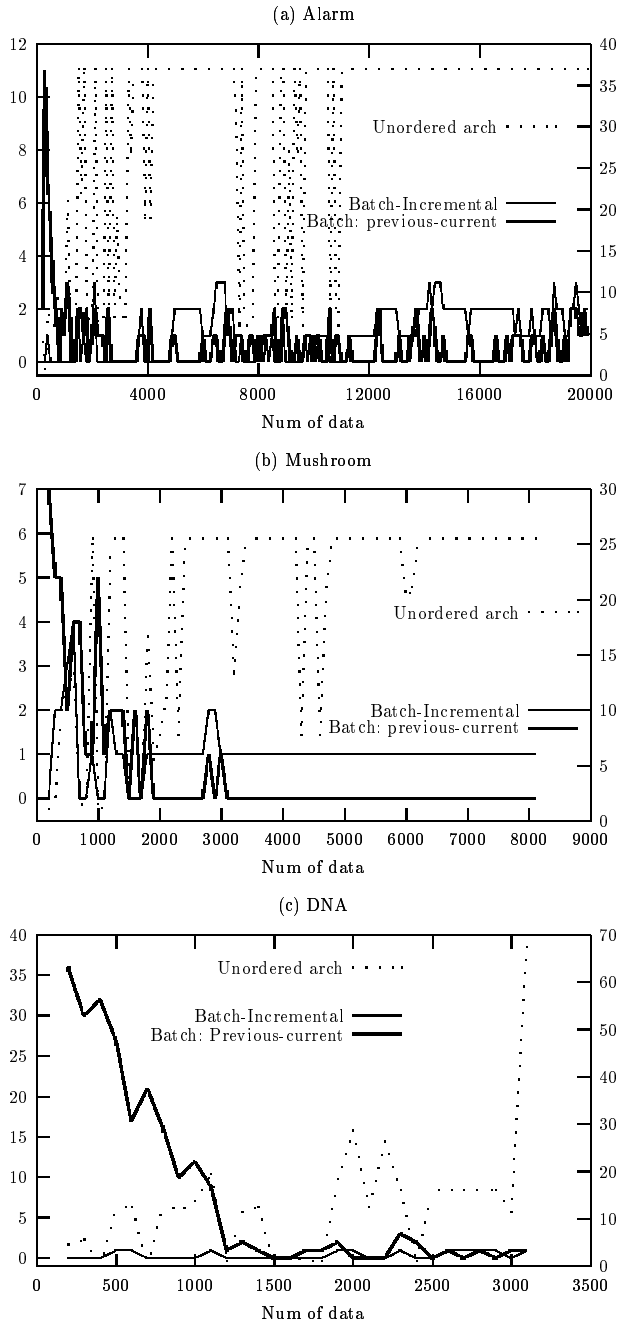
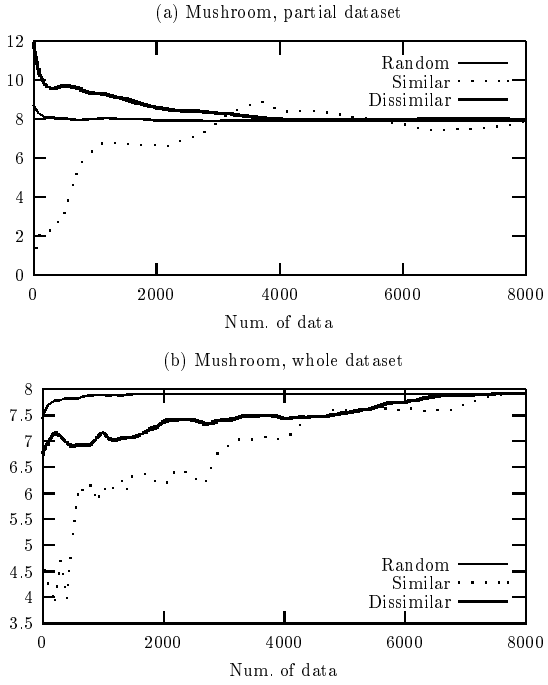


Figure 1. Quality of recovered structures. Each graphic presents three curves; the first (*Unordered arch*), shows the first arch which is not in decreasing cost order. When the number shown coincides with the number of attributes, it means that all arches are in decreasing cost order. The second (*Batch-Incremental*) shows the number of different arches between the trees learnt by the batch algorithm and the ones learnt with the ACO heuristic. The third curve (*Batch: Previous-current*), shows the number of different arches between the former and the current trees learnt by the batch algorithm. Note that the y axis of the first curve is on the right while the y axis of the second and third curves is on the left.

Both graphics, clearly show that when data is presented in random order the intermediate tree structures obtained rapidly tend to the maximum quality obtained when all data is processed. On the contrary, presenting similar instances consecutively intermediate results are the poorest. In between, there are the results obtained when dissimilar data instances are presented consecutively.

Figure 2. Learning curve, quality: $\sum_{i=1}^{n} I(X_{m_i}; X_{m_{j(i)}})$



5 Discussion and Final Conclusions

Previous work on incremental learning of Bayesian Network have focused on learning general network structures, namely, directed acyclic graphs (DAGs). In these works the authors assume that the size of the *sufficient statistics* necessary to recover any possible DAG of the search space is very large and thereof it is not feasible to store them in main memory.

The work proposed by Buntine [2], by Friedman and Goldszmidt [6] and by Lam and Bacchus [8] are similar in the sense that they store the *sufficient statistics* of those structures close to the current one. Friedman and Goldszmidt's system is able to learn networks out of that *sufficient statistics*. In consequence different parts of a Bayesian Network may be learnt with different data instances. Lam and Bacchus' approach is able to incrementally learn a subset of the variables of the network structure.

We presented in this paper an extension of the CL algorithm in order to incrementally learn tree-shaped Bayesian Networks. We proposed the ACO heuristic that guesses when and which part of the structure should be updated. We claim that our algorithm is very reactive, that is, it is able to quickly detect changes in new data and to correctly update the structure. In Figure 1, we could see that the heuristic is sound in the sense that it triggers the updating process only when changes are actually needed. We also noted, that the ACO heuristic does not ensure to provide the same tree as if it was learned with the batch CL algorithm. Even thought, we saw that the learned trees are very close to those obtained with the batch CL algorithm, as seen in Section 4.

The major benefit of our incremental proposition is that it saves computing time. Even when the tree must be updated the number of calculations and the number of comparisons required is very much reduced each time a branch is checked as correctly ordered. The number of comparisons the CL algorithm must perform to order the arches is $\binom{n}{2}+\sum_{i=2}^{n} i(n-i)$, while in our proposition when the first branch is checked as correct the number of comparisons is reduced by $\binom{n}{2}$ and the number of calculations of mutual information is reduced from $\binom{2}{n}$ to a maximum of $\binom{n-1}{2}$. And when the k-th branch is checked as correct, being $1 < k < n$, the number of comparisons is reduced by $k(n-k)$ and the number of tests is reduced from $\binom{n-k}{2}$ to a maximum of $\binom{n-k-1}{2}$. As seen at Figure 1 the tree is not updated most of the times and in very few occasions the first branch is not correctly ordered.

We also want to stress that the learned structures are not biased by the initial ones as we could see in Table 2. There, we observed that the final results are very similar in all the three data instance orders.

ACKNOWLEDGEMENTS

We would like to thank Joan Jou, Ramon Sangüesa and Luis Talavera for a thorough reading of this paper, and to the anonymous referees for many interesting comments.

REFERENCES

[1] I. Beinlinch, G. Suermoundt, R. Chavez, and G. Cooper, 'The ALARM monitoring system', in *European Conference on AI and medicine*, (1989).

[2] W. Buntine, 'Theory refinement on Bayesian networks', in *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, eds., B.D. D'Ambrosio, P. Smets, and P.P. Bonisone, pp. 52–60, (1991).

[3] W. Buntine, 'A guide to the literature on learning probabilistic networks from data', *IEEE Trans. On Knowledge And Data Engineering*, **8**, 195–210, (1996).

[4] C.K. Chow and C.N. Liu, 'Approximating discrete probability distributions with dependence trees', *IEEE Transactions on Information Teory*, **14**, 462–467, (1968).

[5] D.H. Fisher, L. Xu, and N. Zard, 'Ordering effects in clustering', in *Ninth International Conference on Machine Learning*, pp. 163–168, (1992).

[6] N. Friedman and M. Goldszmidt, 'Sequential update of Bayesian network structure', in *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, (1997).

[7] D. Heckerman, 'A tutorial on learning Bayesian networks', Technical Report MSR-TR-95-6, Microsoft Research, Advanced Technology Division, (1995).

[8] W. Lam and F. Bacchus, 'Using new data to refine Bayesian networks', in *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, eds., R. López de Mantaras and D. Poole, pp. 383–390, (1994).

[9] P. Langley, 'Order effects in incremental learning', in *Learning in humans and machines: Towards an Interdisciplinary Learning Science*, eds., P. Reimann and H. Spada, Pergamon, (1995).

[10] Andrew W. Moore and Mary S. Lee, 'Cached sufficient statistics for efficient machine learning with large datasets', *Journal of Artificial Intelligence Research*, **8**, 67–91, (1998).

[11] P.M. Murphy and D.W. Aha. UCI repository of machine learning databases. http://www.ics.uci.edu/ mlearn/MLRepository.html., 1994. Irvine, CA: University of California, Department of Information and Computer Science.

[12] J. M. Peña, J. A. Lozano, P. Larrañaga, and I. Inza, 'Dimensionality reduction in unsupervised learning of conditional gaussian networks.', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **6**(23), 590–603, (2001).