

# Incremental Hill-Climbing Search Applied to Bayesian Network Structure Learning

Josep Roure Alcobé

Escola Universitària Politècnica de Mataró,  
Av. Puig i Cadafalch 101-111, 08303 Mataró, Catalonia, Spain,  
roure@eupmt.es

**Abstract.** We propose two general heuristics to transform a batch Hill-climbing search into an incremental one. Then, we apply our heuristics to two Bayesian network structure learning algorithms and experimentally see that our incremental approach saves a significant amount of computing time while it yields similar networks than the batch algorithms.

## 1 Introduction

Currently, there are many computing learning environments where data records are gathered every day. Organizations today store, in their computer systems, most of their transactions. We could see this organizations as systems whose everyday activity is monitored by means of the data records they produce, and where their state may be represented by knowledge models mined from these data. Knowledge models (or states) may be used to represent the systems themselves, or to explain their behavior, or even to predict their future activities. In this sort of environments, mining algorithms must cope with streams of data that grow continuously.

To mine data-streams, it would be possible to repeatedly use *batch* or *one-shot* algorithms mining old together with new data. This simple approach would spend a great amount of computing time since *batch* algorithms begin from scratch and need to explore the entire data records every time they are used. It would also require a lot of memory space since this simple approach needs to store the whole data. There are real-world environments with strong constraints in computing time and memory space where *incremental* algorithms are best suited. In this sort of environments, *incremental learning* becomes particularly relevant since they are required to use a small constant time per record, to scan datasets only once, to use a fixed amount of memory, and to make a usable model available at any point in time [1]. In this work we introduce two simple and general heuristics in order to obtain incremental algorithms that fulfill the four requirements stated above.

Bayesian networks are models for coping with uncertainty and are based on probability theory. The main advantages of Bayesian networks can be summarized in three points: (i) they allow to express directly the fundamental qualitative relationship of direct causation, (ii) there exists mathematical methods to estimate the state of certain variables given the state of other variables, (iii)

there are methods in order to explain to the user how the system came to its conclusions. Unfortunately, learning Bayesian networks is NP-hard [2].

The rest of this paper is organized as follows. In this section we revise hill-climbing searchers and Bayesian networks in order to give a precise notation. In Section 2, we introduce two heuristics to transform a batch hill-climbing search algorithm into an incremental one. In Section 3, we use our proposal to learn Bayesian network structures incrementally and we show experimental results. Finally, in Section 4, we give some conclusions.

### 1.1 Hill-Climbing Search

The idea of a hill-climbing search algorithm (HCS) is to generate a model in a step-by-step fashion by making the maximum possible improvement in an objective quality function at each step. So, we need to define the following three elements for a given problem: (i) an objective **function**  $S(M, D)$  to measure the quality of a given model  $M$  with respect to a dataset  $D$ , (ii) a set of **traverse operators**  $OP = \{op^1, \dots, op^k\}$  that given an argument,  $A$  and a model  $M$ , obtain a new model  $M' = op^i(M, A)$ , (iii) a **domain**  $\mathcal{MD}$  to define the *legal* models. For example, in the field of Bayesian networks, the domain is the space of DAGs, an operator is adding an arc, and an argument is the added arc. See that a traverse operator  $op^i$  will be used only if it produces a model within the domain  $\mathcal{MD}$ :

**Definition 1 (Neighborhood).** *The neighborhood  $\mathcal{N}(M)$  of a given model  $M$  is the set of all the alternative models that belong to the domain  $\mathcal{MD}$  and that can be built by using a single operator,  $\mathcal{N}(M) = \{M' \mid M' = op^i(M; A) \wedge M' \in \mathcal{MD}\}$*

Note that a neighborhood,  $\mathcal{N}(M)$ , is built using a set of operator and argument pairs,  $\mathbf{OpA}_{\mathcal{N}(M)} = \{(op^i, A_i) \mid op^i(M, A_i) \in \mathcal{D}\}$ .

Now, we are able to describe HCS formally. HCS begins with an initial model  $M_0$  (i.e. the empty model) and iteratively constructs a sequence of models  $M_i$ ,  $i = 0, \dots, n$ , where each model  $M_i$  is the one with the highest score of the models in the neighborhood of  $M_{i-1}$ ,  $M_i = \arg \max_{M \in \mathcal{N}(M_{i-1})} S(M, D)$ . HCS stops when no further improvement can be achieved, that is, when the current model has the highest score of the ones in its neighborhood.

In our incremental approach we will study all the intermediate models traversed by HCS,

**Definition 2 (Search path).** *Let  $M_0$  be an initial model, let  $M$  be the final model obtained by a hill-climbing search algorithm as*

$$M = op_n^{k_n} (\dots (op_2^{k_2} (op_1^{k_1} (M_0; A_1); A_2) \dots; A_n))$$

*The search path is the sequence of operator and argument pairs  $\mathcal{O}_{op} = \{(op_1^{k_1}, A_1), (op_2^{k_2}, A_2), \dots, (op_n^{k_n}, A_n)\}$  used to build  $M$ , or equivalently, the sequence  $\mathcal{M}_M = \{M_0, M_1, \dots, M_n\}$  of intermediate models obtained with the sequence of operators and argument pairs, where  $M_i = op_n^{k_i}(M_{i-1}, A_i)$ .*

Remark that the models in the *search path* are ordered in increasing quality score order,  $S(M_0, D) < S(M_1, D) < \dots < S(M_n, D)$  and that the final model  $M_n$  is a local maxima of the domain  $\mathcal{MD}$  of models.

## 1.2 Bayesian Network Learning

A *Bayesian network* is an annotated directed acyclic graph (DAG) that encodes a joint probability distribution of a set of random variables  $\mathbf{X} = \{X_1, \dots, X_n\}$  each of which has a domain of possible values. Formally, a Bayesian network for  $\mathbf{X}$  is a pair  $S = (B_S, B_P)$  where the first component,  $B_S$ , is a DAG whose vertices correspond to the random variables  $X_1, \dots, X_n$ , and whose edges represent directed dependencies between variables. The set,  $\mathbf{Pa}_i$ , of parents of  $X_i$  is the set of variables with an arc to  $X_i$  in the graph. The model structure yields to a factorization of the joint probability distribution for  $\mathbf{X}$ ,  $P(\mathbf{X}) = \prod_{i=1}^n P(X_i | \mathbf{Pa}_i)$ . The second component,  $B_P$ , represents the set of parameters that quantifies the network. It contains a parameter  $\theta_{ijk} = P(X_i = x_i^k | \mathbf{Pa}_i = \mathbf{pa}_i^j)$  for each possible state  $x_i^k$  of  $X_i$  and for each configuration  $\mathbf{pa}_i^j$  of  $\mathbf{Pa}_i$ .

We will revise two well-known learning algorithms, namely, B [3] and one proposed in [4] that we will call FG. Both are hill-climbing searchers that begin with the arc-less network and for each node perform the operation that most increases the score of the resulting structure and does not introduce a cycle into the network. Algorithms stop when performing a single operation cannot increase the network's score. The difference between B and FG is the neighborhood they use. Algorithm B only uses the addition operator, while algorithm FG uses the addition, reversion and deletion of an arc.

For algorithm B, the neighborhood of a given network structure  $B_S = (\mathbf{X}, E)$  is the set of all network structures that can be obtained from  $B_S$  by adding a single arc  $X_i \rightarrow X_j$  to  $B_S$  such that does not introduce a cycle,

$$N_B(B_S) = \{ (\mathbf{X}, E \cup \{X_i \rightarrow X_j\}) \mid E \cup \{X_i \rightarrow X_j\} \text{ is a DAG} \}$$

while for algorithm FG, the neighborhood of a given network structure,  $B_S$ , is the set of all networks that can be obtained from  $B_S$  by adding, reversing or deleting a single arc  $X_i \rightarrow X_j$  to  $B_S$  such that does not introduce a cycle,

$$N_{FG}(B_S) = \{ (\mathbf{X}, E') \mid E' = E \cup \{X_i \rightarrow X_j\} \text{ is a DAG} \\ \cup \{ (\mathbf{X}, E') \mid E' = E - \{X_i \rightarrow X_j\} \text{ is a DAG} \}$$

There are basically two approaches to the scoring metrics in the field of Bayesian network structure, one is the Bayesian Dirichlet (BDeu) approach, see [2], and the other is the Minimum Description Length (MDL) approach, see [5].

## 2 Incremental Hill-Climbing Search

In this section we propose two general heuristics in order to transform any batch Hill-Climbing searcher into an incremental one. Before doing so, we analyze the HCS's search space.

### 2.1 Search Space in Incremental Learning

The search space of knowledge models can be described [6] as an n-dimensional space whose shape is given by a quality scoring function  $f$ . The aim of the search strategy is to reach the point (i.e. the knowledge model) with the highest  $f$  score. In off-line environments, the function  $f$  is static and thus the shape of the surface is constant. On the contrary, in online environments each new instance changes the form of  $f$  and modifies the contours of the surface.

Incremental algorithms are sensitive to the order in which data instances are presented to the algorithm [6], as a consequence of changing the contours of the score function  $f$ . That is, given two sample orders,  $O_1$  and  $O_2$ , of a database  $D$ , an incremental algorithm may output different domain models when fed with order  $O_1$  or with order  $O_2$ . Ordering effects are due to the nature of the incremental processing of data combined with the tendency of hill climbing methods to get stuck at local maxima. When dissimilar instances are consecutively presented, results are much better than when similar instances are presented successively [7]. This occurs because, in the former case, initial observations are sampled from different parts of the description space leading initial models to approximate the actual probability distribution of the dataset, while in the later, rather skewed models may be built at the beginning, biasing the rest of the learning process.

At this point of the discussion, we want to stress that one would prefer search space surfaces that do not change drastically when few data are added to the current dataset or when the newly gathered data slightly modifies the underlying probability distribution. Thereof, we prefer quality functions that are somehow *continuous* over the space of datasets. That is, quality functions that given a knowledge model and two arbitrarily close datasets, they return two arbitrarily close quality values. We can measure the *closeness* of two datasets as the Kullback-Leibler divergence,  $D_{KL}(D||D') = \log(P(D)/P(D'))$ . Remark that *continuous* quality functions will usually be preferred since it is required that models are not scored very differently when they are evaluated with respect to similar datasets. For example, we do not want quality functions to be sensitive in the presence of few noisy instances in a dataset.

Now we will present two heuristics in order to obtain incremental hill climbing searchers. Noting that the search path is an ordered sequence of a subset of models in the search space, the heuristics assume that if the order of the models in the search path change when new data are gathered, it means that the new data significantly change the underlying probability distribution. In other words, that the new data have significantly changed the surface of the search space.

## 2.2 Traversal Operators in Correct Order

We call the first heuristic *Traversal Operators in Correct Order*, (TOCO). This heuristic assumes that it is worth to update an already learned model only when the new data alter the order of the models present its learning path.

In our incremental approach, we will keep the search path of the former learning step. If, when new data instances are available, the order of models in the path changes, we can conclude two facts. First, since we assume that the quality function is *continuous*, we know that  $P(D)$  and  $P(D \cup D')$  are significantly different and thus, that the newly available instances provide new information. Second, we know that HCS would follow a different search path through the space of models and possibly obtain a different one. On the contrary, when the order still holds we conclude that  $P(D)$  and  $P(D \cup D')$  are very similar and thereof, we assume that HCS would follow again the same path and obtain the same model. Thus, in the former case we trigger the HCS algorithm while in the latter we do not revise the structure.

When the order of models in the path changes, we will assume that the model built by means of the correctly ordered traverse operators is still correct and thus we will use it as the initial model for the HCS algorithm. So, the benefit of the TOCO heuristic is twofold. First, the model will only be revised when it is *invalidated* by new data, and second, in the case that it must be revised, the learning algorithm will not begin from scratch. Formally,

**Definition 3 (TOCO heuristic).** Let  $D$  be a dataset,  $M$  be a model,  $S(M, D)$  be a scoring metric, HCS a hill-climbing searcher,  $\{op^1, \dots, op^k\}$  be a set of traverse operators which given an argument  $A$  and a model  $M$  return a new model  $M'$ . Also, let  $\mathbf{OpA}_{\mathcal{N}(M)}$  be the set of operators and argument pairs with which neighborhood  $\mathcal{N}$  of model  $M$  is obtained. Let  $M$  be a model learned by HCS where  $M = op_n^{k_n}(\dots(op_2^{k_2}(op_1^{k_1}(M_0, A_1), A_2) \dots, A_n))$  is built with the search path,  $\mathcal{O}_{Op} = \{(op_1^{k_1}, A_1), \dots, (op_n^{k_n}, A_n)\}$  where

$$\forall i \in [1, n] : (op_i^{k_i}, A_i) = \arg \max_{(op_i^{k_i}, A) \in \mathbf{OpA}_{\mathcal{N}(M_{i-1})}} S(op_i^{k_i}(M_{i-1}, A), D)$$

Let  $D'$  be a set of new data, the TOCO heuristic states that HCS learns a new model  $M' = op_{n'}^{k_{n'}}(\dots(op_{2'}^{k_{2'}}(op_{1'}^{k_{1'}}(M_{ini}, A_{1'}), A_{2'}) \dots, A_{n'})$  corresponding to  $D_t = \{D \cup D'\}$  where traverse operators and arguments are obtained as before

$$\exists i \in [1; n'] : (op_i^{k'_i}, A_i) = \arg \max_{(op_i^{k'_i}, A) \in \mathbf{OpA}_{\mathcal{N}(M_{i-1})}} S(op_i^{k'_i}(M_{i-1}, A); D_t)$$

and where the initial model is  $M_{ini} = op_j^{k_j}(\dots(op_2^{k_2}(op_1^{k_1}(M_0, A_1), A_2) \dots, A_j))$  where  $(op_j^{k_j}, A_j)$  is the last operator and argument pair which is in correct order, that is, the last pair that produces the model with the highest score among pairs in the search path  $\mathcal{O}_{Op}$

$$(op_{j+1}^{k_{j+1}}, A_{j+1}) \notin \arg \max_{(op_i^{k'_i}, A) \in \mathbf{OpA}_{\mathcal{N}(M_{i-1})} \cap \mathcal{O}_{Op}} S(op_i^{k'_i}(M_{i-1}, A); D_t):$$

Note, that there are two extreme cases. On the one hand, when  $j = 0$ , it means that the first operator and argument pair is not in correct order and thus the new revised structure will be learned from scratch. On the other, when  $j = n$ , it means that all pairs were in correct order and thus the current structure does not need to be updated.

### 2.3 Reduced Search Space

The second heuristic applies when the current structure needs to be revised. Basically, it reduces the search space by avoiding to explore those parts of the space where low quality models were found during former search steps. We call this heuristic Reduced Search Space, RSS from now on.

Given a model  $M$ , HCS scores operator and argument pairs  $\mathbf{OpA}_{\mathcal{N}(M)}$  in order to obtain the best model of the neighborhood  $\mathcal{N}(M)$ . During this process, the RSS heuristic stores, into a set  $\mathcal{B}$ , the  $nRSS$  pairs closest to the best one. So, in the next search steps, when new data are available, HCS restricts the

search to the models of the neighborhood obtained with these pairs, namely  $\text{OpA}_{\mathcal{N}(\mathcal{M})} \cap \mathcal{B}$ . The rationale for this is that if the score of a model is very low it will not grow very much with a short chunk of new data instances. Note that when  $nRSS$  is close to zero the search space is very much reduced and thus the model obtained may be biased towards the current one and it may substantially differ from the one obtained using the whole search space.

See that this heuristic is banked again on the fact that the surface of the search space does not change drastically when the data underlying probability distribution changes only slightly. If it did, the *incremental* algorithm would not be able to reach models of high quality since they might fall out of the  $\mathcal{B}$  sets.

## 2.4 Incremental hill-climbing search

Now we are ready to show the incremental hill-climbing search (iHCS) which incorporates the TOCO and RSS heuristics. Algorithm iHCS, see Algorithm 1, performs two different tasks. The first one corresponds to the TOCO heuristic and consists in checking whether the learning path of the previous learning step are in correct order and to state the initial model,  $M_{ini}$ , from which the search will be resumed. The second task consists in performing the search (i.e. update the model) using the newly available data.

---

### Algorithm 1 Incremental Hill-Climbing Search

---

**Require:** a domain of models  $MD$ , a set of operators  $OP = fop^1; \dots; op^k g$ , a database  $D$ , a scoring function  $S(M; D)$ , the search path of the last learning step  $O_{Op}$  used to obtain the former model  $M_{for}$ , the sets  $B_i$  for each variable, two positive integers  $nRSS$  and  $k$  to state the number of pairs in  $B$  to consider

**Ensure:**  $M$  be a model of high quality in  $MD$

let  $(op_j^{k_j}; A_j)$  be the last ordered pair in  $O_{Op}$

$M_{ini} = op_j^{k_j} (\dots (op_2^{k_2} (op_1^{k_1} (M_0; A_1); A_2) \dots); A_j)$

$(M_{ini} = M_{for})?$  Use  $nRSS$  : Use  $nRSS + k$

$i = 0; M_i = M_{ini}$

**repeat**

  oldScore =  $S(M_i; D)$ ;  $i = i + 1$

$M_i = op_i^{k_i} (M_{i-1}; A_i)$  where  $(op_i^{k_i}; A_i) = \arg \max_{(op^k, A) \cap B_i} S(op_i^{k_i} (M_{i-1}; A_i); D)$

**if**  $M_{ini} \notin M_{for}$  **then**

    Calculate  $B_i$  for each pair in  $O_{Op}$

**end if**

**until** oldScore  $\geq S(M_i; D)$

---

Note that although all the operator and argument pairs are in correct order the incremental algorithm will try to improve the model obtained in the previous learning step. It is convenient to allow iHCS algorithm to use new operators as more data are available because more complex structures (i.e. higher degree relations among variables) can reliably be evaluated. So, there are two different situations (i.e. *pre-conditions*) in which the iHCS algorithm gets to the *repeat* statement. The first one happens when the TOCO heuristic finds an operator and argument pair incorrectly ordered ( $M_{ini} \neq M_{for}$ ) and, in consequence,

the iHCS fires the revising process. The second one happens when the TOCO heuristic finds all operators in correct order ( $M_{ini} = M_{for}$ ) and thus the former model is still *valid* and the iHCS algorithm tries to use new operators just in case that the former model can be improved. We will use the RSS heuristic differently in these two situations. In the first one, since the new data provoke the revision of the current model, the RSS heuristic should not reduce very much the search over the space of models. On the contrary, in the second situation, since the new data have not *invalidated* the current model it suffices to try those operator and argument pairs that were most promising in the former learning steps. Another difference is that in the first situation the sets of the RSS heuristic,  $\mathcal{B}$ , are not updated, while in the second the RSS sets are updated since  $D_{KL}(D||D \cup D')$  is big enough to provoke changes in the ranking of operator and parameter pairs.

### 3 Incremental BN structure learning

In this section we use our TOCO and RSS heuristics to obtain incremental version of algorithms B and FG. Before doing so we want to note that both MDL and BDeu quality measures used in the field of Bayesian networks are *continuous* in the sense that we introduced in Section 2.1 and thereof that our heuristics may work well. See that the MDL measure is a decreasing function of the Kullback-Leibler divergence between the probability distribution,  $P(D)$ , of a dataset,  $D$ , and the probability distribution,  $P(B_S)$ , of a Bayesian network structure  $B_S$  [5], from where it is straightforward to show its *continuity* and also see that under some conditions  $MDL(B_S, D) = BDeu(B_S, D) + \mathcal{O}(1)$  [8].

#### 3.1 Incremental B and FG

Here, we adapt the TOCO and RSS heuristics to B and FG algorithms. In order to use our TOCO heuristic we keep the order in which the operators are used (the search path) to built the structure when dataset  $D$  is available. In the following learning steps, when new data  $D'$  are presented, this order is checked. If the order does not hold at a given point of the path when the operator and argument pairs are measured with respect to the old and new data  $D \cup D'$ , the search path is resumed from that point. We say that the order does not hold when there exists one operator in the order that does not maximize the score. More precisely, let  $(op^i, (X_m, X_n))_k$  be an operator  $op^i$  and an arc  $(X_m, X_n)$  in position  $k$  within the order  $\mathcal{O}$ , we say that  $(op^i, (X_m, X_n))_k$  is not in correct order when exists another operator  $(op^j, (X_p, X_q))_r$  in the order where  $k < r$  and such that  $S((op^i(B_S, (X_m, X_n)), D \cup D') < S((op^j(B_S, (X_p, X_q)), D \cup D')$  and the score  $S$  is calculated with both the old and new data instances.

We experimentally saw that this condition of order among arcs was too strong. That is, in many cases the order of operator and argument pairs did not hold and when the learning process was fired, it yielded the same set of operator and argument pairs but in a different order. For this reason, we relax this condition by introducing a *window* when checking the order. That is, when new data is available we do not require the best pair to be exactly at the same place in the order but in a place nearby. In this way, the  $k$ -th pair in the order  $\mathcal{O}$  will be considered correct if taking into account the score calculated with the

**Table 1.** Results with B ( $nTOCO=2$ ,  $nRSS=2$ ) and FG ( $nTOCO=3$ ,  $nRSS=2$ )

Algorithm B					Algorithm FG					
	S i/b	Str. Eq.	Non. Fired%	TG%	CG%	S i/b	Str. Eq.	Non. Fired%	TG%	CG%
<b>Synthetic. #Arcs Batch: 61</b>					<b>Synthetic. #Arcs Batch: 61</b>					
Rnd	0.987	40.00(16.25)	79.66(3.58)	75.46	77.69	0.985	31.80(5.90)	96.40(3.68)	84.14	84.80
Sim	0.985	36.60(8.32)	61.46(4.90)	64.26	68.98	0.987	31.00(15.36)	82.06(23.69)	81.16	83.03
Dis	0.984	38.60(17.81)	55.13(7.49)	65.02	70.64	0.990	41.00(16.31)	99.39(2.07)	71.77	77.05
<b>Adult. #Arcs Batch: 22</b>					<b>Adult. #Arcs Batch: 22</b>					
Rnd	0.999	13.80(9.10)	96.39(2.42)	85.20	78.87	1.031	11.00(6.16)	99.39(2.07)	96.87	84.31
Sim	1.001	12.40(2.28)	94.58(1.53)	86.08	78.56	1.014	9.00(4.47)	99.14(0.67)	94.53	83.70
Dis	1.000	11.60(7.82)	97.17(0.61)	87.31	78.27	1.042	6.60(3.35)	99.92(0.37)	96.59	86.69
<b>Alarm. #Arcs Batch: 53</b>					<b>Alarm. #Arcs Batch: 54</b>					
Rnd	1.002	42.20(9.53)	68.69(4.90)	75.00	73.97	1.002	40.40(7.69)	95.18(1.83)	92.29	88.88
Sim	1.002	39.20(8.53)	24.24(6.66)	39.00	44.39	1.003	45.60(7.29)	74.07(3.73)	78.22	74.49
Dis	1.001	38.00(8.60)	51.82(3.61)	58.91	65.79	1.001	39.60(5.76)	92.26(5.35)	87.89	86.61
<b>Mushroom. #Arcs Batch: 71</b>					<b>Mushroom. #Arcs Batch: 71</b>					
Rnd	0.998	52.60(4.15)	28.00(12.82)	47.03	51.06	1.004	45.60(15.91)	73.83(9.63)	75.02	66.90
Sim	1.000	58.20(17.91)	11.25(11.73)	15.32	24.65	1.002	51.00(7.35)	36.79(9.94)	39.67	42.16
Dis	1.000	66.60(13.31)	10.75(5.48)	17.62	31.67	1.005	56.20(18.89)	47.41(7.92)	50.36	51.70
<b>Nursery. #Arcs Batch: 11</b>					<b>Nursery. #Arcs Batch: 8</b>					
Rnd	1.000	9.40(4.60)	97.97(1.78)	73.24	68.57	1.006	3.20(0.89)	100.0(0.00)	91.70	76.28
Sim	1.000	10.80(0.89)	62.66(8.15)	56.03	54.12	1.002	4.40(4.38)	96.12(3.64)	76.86	70.47
Dis	1.000	8.80(5.18)	82.34(12.18)	64.72	61.96	1.002	6.00(2.00)	98.91(0.85)	83.58	72.62

new data  $D \cup D'$ , it is between the  $(k - nTOCO)$ -th and the  $(k + nTOCO)$ -th position, where  $nTOCO$  is a parameter. The intuition is that we need more evidence that an arc is incorrect to revise the structure.

In order to introduce the RSS heuristic into algorithms, we keep for each variable as many lists of candidates as parents are introduced for the variable. So, when a revision of the parents of a variable is needed, the algorithm will search only among those variables that belong to the list of candidates. The number of variables to be stored into the lists in order to restrict the search is specified by means of a parameter  $nRSS$ .

### 3.2 Experimental Results

In this section we compare the performance of repeatedly using the batch algorithms against the corresponding incremental approach. We used the datasets Adult(48.842 instances and 13 variables), Mushroom (8.124 inst. and 23 var.) and Nursery (12.960 inst. and 9 var.) from the UCI machine learning repository [9], the Alarm dataset (20.000 inst. and 37 var.) [2] that is a standard benchmark in the Bayesian network literature and a synthetic dataset that were kindly donated by Robert Castelo [10], Synthetic (100.000 inst. and 25 var.).

We fed data instances to both algorithms in chunks of 100 and we compared the Bayesian Networks yielded during all the learning process. We also presented data instances in different orders to test the behavior of our incremental approach. 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, and the results presented in this section correspond to the mean and the standard deviation of the quantity being analyzed.

The main objective of our incremental algorithms is to reduce the time spent in learning a new network structure when the system already learned one from

past data and still produce structures of high quality. Table 1 shows the results obtained with algorithms B and FG and their corresponding incremental approaches. This table shows for the three data orders and for each dataset the following measures: Column “**S i/b**” shows the score of the networks produced by the incremental approach divided by the score of networks produced by the batch approach. When this column takes values larger than one, favor the batch approaches and vice versa. Column “**Str. Eq.**” show the structural equalities between the Bayesian networks obtained with both approaches. Namely the number of arcs shared by the structures being compared. See also that the head of each dataset, in the table, shows the number of arcs that the structures of the batch approach have. Column “**Non Fired%**” shows the percentage of the times that the TOCO heuristic does not fire the revising process. Columns “**TG%**” and “**CG%**” show the time gain and the gain in the number of calls to the scoring function performed by the algorithms, respectively. The columns are calculated as  $(1 - nI/nB) \times 100$ , where  $nI$  and  $nB$  are the number of clock ticks or calls performed by the incremental and the batch approach respectively. The number of calls are a good measure of the time spent because of three reasons. Firstly, the measures are expensive in time, secondly, the number of calls gives an idea of the number of the different network structures being considered by the search algorithms, and thirdly it is decoupled from the computer architecture.

We can see in Table 1 that for both algorithms, the incremental approaches save a significantly amount of CPU clock ticks while the quality of the final Bayesian networks is very close to the ones obtained with the batch approaches. See also that the Mushroom dataset is generally the most *difficult* to learn incrementally in the sense that incremental algorithms obtain the lowest time gain. This may be due to the fact that there are many arcs that bring similar quality to the network as can be seen in Table 1 where there are few structural equalities (Str. Eq.) and the quality ratio (S i/b) is still close to 1, and thus the order of the operator and argument pairs vary very much with new data instances.

We can also see that the benefit of using our TOCO heuristic is twofold. First, it avoids firing the revising process when it is not necessary (see column “Non Fired%”) and second, it does not begin from scratch when the revision process is fired. We observed that the mean is to begin the revising process at the middle of the searching path. We can see the benefit of our RSS heuristic noting that even when the revising process is fired often the gain in number of calls to the scoring functions is still high. Remember that when the revising process is fired the search space is smaller than the one used by the batch algorithms.

We want to remark that our incremental approaches do not suffer from ordering effects, that is, they are able to recover from biased past Bayesian networks yielding final structures of almost the same quality as the batch approaches do, independently of the order in which data are presented to the learning algorithms. As seen in the table, incremental algorithms spend more time processing orders where similar instances are presented consecutively since the data distribution changes and networks must be updated often. This results show that our TOCO heuristic is very good in detecting changes in data distribution.

## 4 Discussion and Final Conclusions

There are few works on incremental Bayesian network learning and here we will briefly comment the most relevant from our viewpoint. Buntine [3] stores in memory the most promising networks and restricts the search among them when new data is available performing a sort of beam search. Friedman et al. [4] and Hulten et al. [11] use *two-way* operators (i.e. add, delete and reverse arcs) to obtain a new network from the current one. These operators allow them to reconsider the arcs added during the past learning steps and to perform a sort of back-tracking to revise the current structure. We also conducted experiments with this approach that showed that our heuristics obtained structures of higher quality at the expense of a slightly lower time gain.

We also want to comment that in order to avoid processing data multiple times we stored *sufficient statistics* in memory using AD-tree structures [12]. The main drawback of this approach is that the memory required grows exponentially with the number of attributes. We reduced this problem in our implementation, however, the full discussion of our approach is beyond the scope of this paper.

As conclusion, we would like to remark that TOCO and RSS heuristics are a general method that allows to transform any batch hill climbing searcher into an incremental one that fulfills the four constraints introduced in Section 1.

### Acknowledgments

We would like to thank to Ramon Sangüesa and Luis Talavera for their insightful comments. Work supported by FEDER and CICYT, TIC-2003-083282-C05-02

### References

1. Domingos, P., Hulten, G.: A general framework for mining massive data streams. *Journal of Computational and Graphical Statistics* **12** (2003)
2. Cooper, G., Herskovits, E.: A Bayesian method for the induction of probabilistic networks from data. *Machine Learning* **9** (1992) 309{347}
3. Buntine, W.: Theory refinement on Bayesian networks. In D'Ambrosio, B., Smets, P., Bonisone, P., eds.: *Proceedings of the 7th UAI*. (1991)
4. Friedman, N., Goldszmidt, M.: Sequential update of Bayesian network structure. In: *Proceedings of the 13th UAI*. (1997)
5. Lam, W., Bacchus, F.: Learning Bayesian belief networks. an approach based on the MDL principle. *Computational Intelligence* **10** (1994) 269{293}
6. Langley, P.: *Elements of machine learning*. Morgan Kaufmann (1996)
7. Fisher, D., Xu, L., Zard, N.: Ordering effects in clustering. In: *Ninth International Conference on Machine Learning*. (1992) 163{168}
8. Bouckaert, R.: *Bayesian belief networks: from inference to construction*. PhD thesis, Utrecht University (1995)
9. Murphy, P., Aha, D.: UCI repository of Machine Learning databases. <http://www.ics.uci.edu/mllearn/MLRepository.html>. (1994)
10. Castelo, R., Koeka, T.: On inclusion-driven learning of Bayesian networks. Technical Report UU-CS-2002-05, University of Utrecht (2002)
11. Hulten, G., Domingos, P.: Mining complex models from arbitrarily large databases in constant time. In: *Proceedings of the Eighth ACM SIGKDD*. (2002)
12. Moore, A.W., Lee, M.S.: Cached sufficient statistics for efficient machine learning with large datasets. *Journal of A.I. Research* **8** (1998) 67{91}