Journal of Scientometric Research, 2024; 13(3):745-756.
https://www.jscires.org

Research Article

# AMBV: An Optimized Generic Viterbi Algorithm for Bayesian Networks

Pierre-Samuel Gréau-Hamard[1,2,*], Moïse Djoko-Kouam[1,2], Yves Louët[2]

[1]Informatics and Telecommunications Laboratory, ECAM Rennes Louis de Broglie, Bruz, FRANCE.
[2]Institut d'Electronique et des Technologies du numéRique (IETR), CentraleSupélec, Rennes, FRANCE.

## ABSTRACT

Bayesian Networks is a family of machine learning models widely used in various applications such as speech recognition, Protocol Reverse Engineering, or more generally the retrieval of hidden information in data. These models, at the crossroads of probability theory and graph theory, allow intuitive modelling and simple interpretation of the results. In this paper, we are interested in inferring the most probable state of a discrete Bayesian network. In the case of a Hidden Markov Model, the Viterbi algorithm is usually used. However, although accurate, it is not optimized, and when generalized to any number of variables per time slice, its complexity increases exponentially. This is why we have developed an optimized version of the Viterbi algorithm, Automatic Markov Boundaries construction optimized Viterbi (AMBV), taking advantage of the fact that once a model is trained, it is usually used to analyze many observations. Moreover, in case of sparse probability distributions of the variables, an additional level of optimization is used. Finally, in order to make possible the inference of the most probable state of any discrete Bayesian network, a mechanism for automatically generating a set of Markov Boundaries for the network has been proposed. We will show that AMBV performs significantly better than the classical Viterbi algorithm as soon as the complexity of the network increases sufficiently.

**Keywords:** Viterbi, Bayesian Networks, Optimized, Generic, Most probable network state, Automatic Markov Boundaries, Sparse probability distributions.

**Correspondence:**
**Pierre-Samuel Gréau-Hamard**
Informatics and Telecommunications Laboratory, ECAM Rennes Louis de Broglie, Bruz, FRANCE.
Email: greauhamard.pierresamuel@gmail.com

**Received:** 23-07-2022;
**Revised:** 21-11-2022;
**Accepted:** 03-08-2023.

## INTRODUCTION

Bayesian Networks (BNs) is a family of mathematical models at the crossroads of probability and graph theory. Their use extends to many fields, such as genetics, biomonitoring, document classification, speech or image recognition. For example, they are used in[1] to generate Gene Regulatory Networks from gene expression observations. They are also found in,[2] where the authors propose to analyse the sentiment about an election in posts on social media like Twitter. Paper[3] also exposes how, associated to deep neural networks, they can be used to perform Voice Activity Detection. In,[4] the authors use the Viterbi algorithm to find the most probable sequence of states of a Compound Hidden Markov Model, in order to identify human activities from cameras and depth captors' data. Lastly, paper[5] makes use of an adaptive Viterbi algorithm to perform image segmentation with the help of an adaptive Hidden Markov Model.

**DOI:** 10.5530/jscires.20040285

They allow to easily model a system through the probabilistic interactions between the different variables characterizing it. It is then possible, if necessary, from observations, to learn the different conditional probability distributions of the system variables.[6] It is also possible to infer the joint probabilities of each variable or group of variables, as well as the most probable global state of the network.

In this paper, we are interested in the latter inference applied to discrete BNs. If the reader is interested in the other types of inference, the paper[7] constitutes a good starting basis. In the case of Hidden Markov Models (HMMs),[8] one of the simplest and most widely used BNs, this problem is solved by the Viterbi algorithm.[9] However, HMMs, containing only two variables per time slice, one hidden and one observed, are only a special case of Dynamic Bayesian Networks (DBNs),[10] in which the number of variables per time slice is arbitrary, and in which each variable is likely to be observed or not at each instant. The Viterbi algorithm is therefore not applicable in its original form, but its principle is nevertheless extensible to DBNs by considering the global state of all the variables of a time slice as the hidden state, and by checking that the considered states are compatible with the observation. For a more in-depth approach to DBN,[7] is a reference in the domain.

In turn, DBNs can be seen as a special case of BNs containing a cyclic repetition of a pattern where each variable has a single conditional probability distribution common to all patterns. It is therefore once again possible to apply the principle of the Viterbi algorithm, but it is necessary to define beforehand the different Markov Boundaries (MBs) describing the network, as they are no longer directly equivalent to time slices. The changes mentioned earlier for the application to DBNs must also be used here.

We called the algorithm resulting from our suggestions Automatic Markov Boundaries construction optimized Viterbi (AMBV). However, AMBV, like Viterbi, has a complexity that increases exponentially with the number of links between variables and linearly with the number of variables and inferences to be made. To partially overcome this problem, we have explored two optimization directions: the precomputation of all the quantities that do not depend on the observations, and a mechanism that allows us to ignore the transition subsets from a state of a Markov Boundary to a state of the next Markov Boundary whose probability are zero. This allows us to greatly reduce the complexity when a large number of inferences are made with the same network, and/or the probability distributions of the variables are relatively sparse. On the other hand, for simple networks, the large overhead required makes AMBV much slower than a simple unoptimized Viterbi extended to BNs.

The rest of this paper is organized as follows: in section II, discrete Bayesian Networks are briefly introduced; in section III, we present the basic Viterbi algorithm extended to Bayesian Networks that we use as a reference; in section IV, we describe the proposed algorithm, AMBV; in section V, we experimentally compare AMBV to the state of the art; and finally, we conclude in section VI.

## DISCRETE BAYESIAN NETWORKS

Bayesian networks are a family of graphical models that represent a system through the probabilistic links between its different characteristic variables. The nodes of the graph represent the variables, while the directed arcs represent the conditional links between variables. The interest of this network lies in the fact that it gives the possibility to infer at low cost any probabilistic relationships between the variables.

In a Bayesian network, any conditional relationship is possible, as long as no cyclic relationship appears, i.e. no variable depends directly or indirectly on itself.

The construction of such a network takes place in four steps. Firstly, the characteristic random variables of the system are identified. Then, the number of possible states for each variable is defined. Afterwards there is the specification of the conditional links between variables. This step, unlike the two previous ones, can be done in two ways: manually, if the relationships between variables are known a priori, as it is the case in this article, or automatically, by learning, if it is not the case. Finally, the last step consists in specifying the exact parameters of the probability tables for each of the variables. Generally, and it is the case in this article, all or part of the distributions of the variables are unknown, and this is why a learning process is set up in order to find parameters allowing to explain the observations of the system as well as possible.

There are three types of variables in a BN:

- Input variables, which are controlled and observable
- Output variables, which are not controlled, but are observable
- Hidden variables, which are neither controlled nor observable

Let's take the example of Figure 1. This network has four variables: a, b, c, and d. The variable a is an input variable, having three possible states: $a_0$, $a_1$, and $a_2$. The variables b and c are hidden and can take two states: $b_0$ and $b_1$, $c_0$ and $c_1$, respectively. d is an output variable, having two possible states: $d_0$ and $d_1$. b is conditionally dependent on a, and d is conditionally dependent on b and c. From a graphical point of view, it can be said that b is a child of a or a parent of b, and that d is a child of b and c, or b and c are parents of d. This means that the probability distributions explicitly included in the network are: P(a), P(b|a), P(c), and P(d|b,c). A possible representation of the probability tables is also given in Figure 1.

Bayesian networks have a fundamental property, allowing inference and learning at low cost: conditional independence. The independence between two variables conditional on a third one, for example a and d conditionally on b in our network, translates into the fact that if the state of b is known, then the state of a has no influence on d and vice versa; this property allows us to consider independently the variables (a, b) and (b, d), and thus to reduce the complexity of the calculations.

## VITERBI INFERENCE

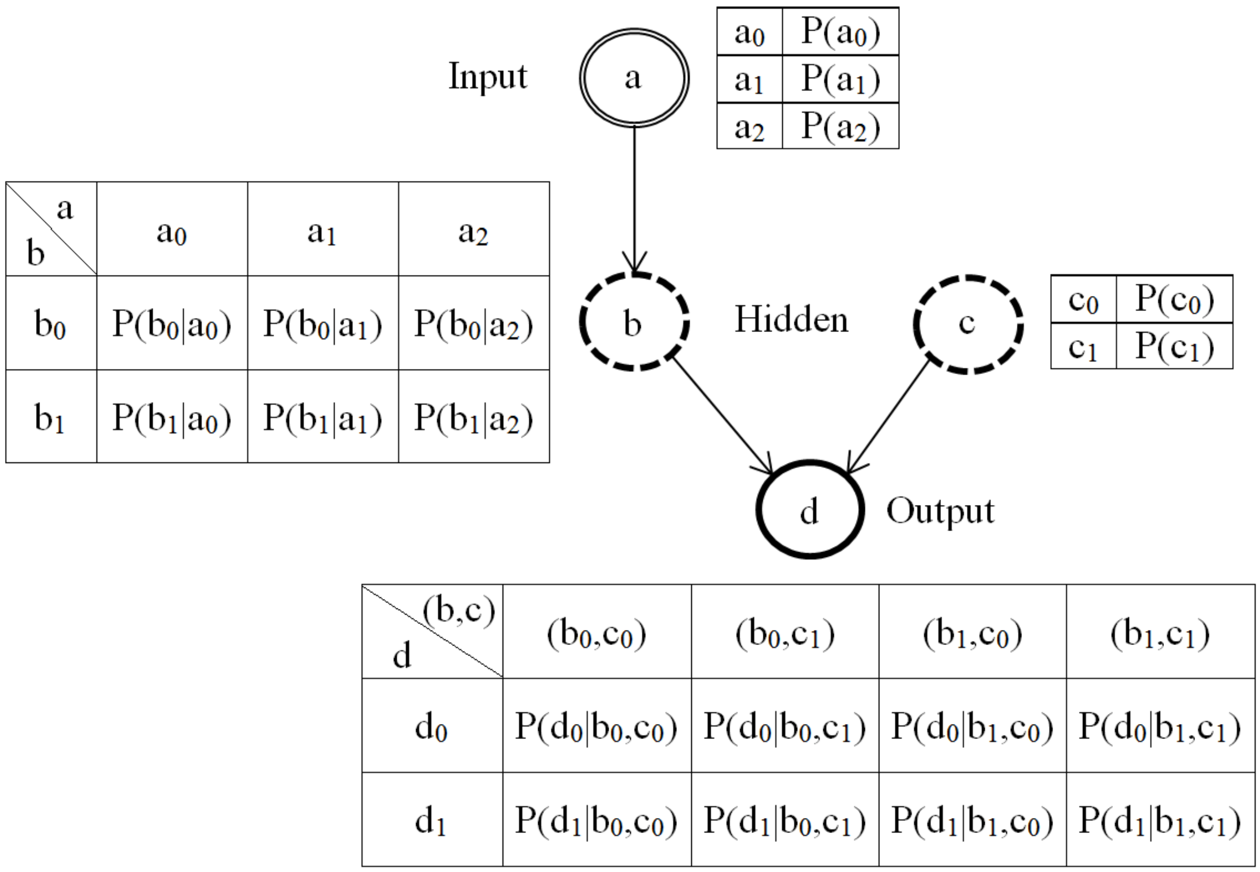
In this section, we present the Viterbi algorithm extended to work on any BN we use as a comparison reference.

### Markov Boundaries

The Viterbi inference is based on the conditional independence property in Bayesian networks. In this context, let us define the notion of Markov Boundary of a set of variables of the network as the minimal set of variables of the network necessary to completely define the considered variables.

Practically, the Markov Boundary corresponds to the parents of the considered variables, their children, and the parents of their children. In the example in Figure 2, the Markov Boundary for Variable B would consist of Variables A (parent and parent of child), D (child), F (child), and C (parent of child).

Now that the notion of Markov Boundary is established, it will be used in Viterbi inference to compute the most probable state



| | |
|---|---|
| $a_0$ | $P(a_0)$ |
| $a_1$ | $P(a_1)$ |
| $a_2$ | $P(a_2)$ |

| b \ a | $a_0$ | $a_1$ | $a_2$ |
|---|---|---|---|
| $b_0$ | $P(b_0 \| a_0)$ | $P(b_0 \| a_1)$ | $P(b_0 \| a_2)$ |
| $b_1$ | $P(b_1 \| a_0)$ | $P(b_1 \| a_1)$ | $P(b_1 \| a_2)$ |

| | |
|---|---|
| $c_0$ | $P(c_0)$ |
| $c_1$ | $P(c_1)$ |

| d \ (b,c) | $(b_0,c_0)$ | $(b_0,c_1)$ | $(b_1,c_0)$ | $(b_1,c_1)$ |
|---|---|---|---|---|
| $d_0$ | $P(d_0 \| b_0,c_0)$ | $P(d_0 \| b_0,c_1)$ | $P(d_0 \| b_1,c_0)$ | $P(d_0 \| b_1,c_1)$ |
| $d_1$ | $P(d_1 \| b_0,c_0)$ | $P(d_1 \| b_0,c_1)$ | $P(d_1 \| b_1,c_0)$ | $P(d_1 \| b_1,c_1)$ |

**Figure 1:** An example of Bayesian Network.

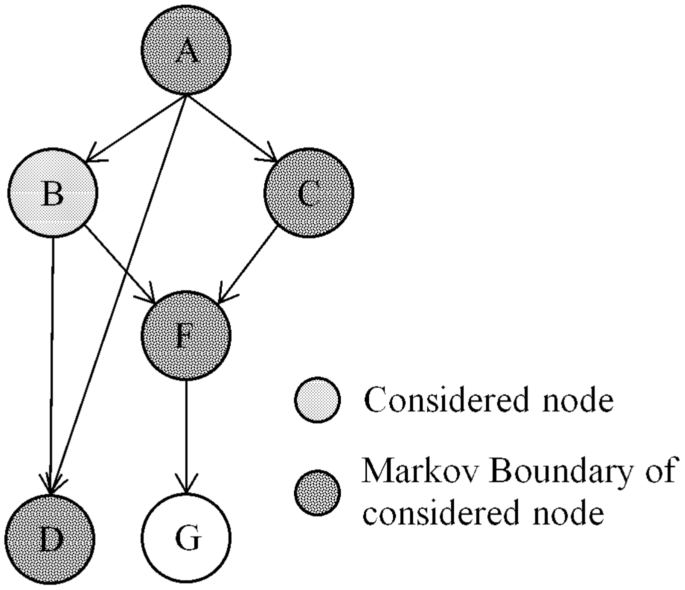


**Figure 2:** Markov Boundary determination example.

of the network. The general principle is to recurrently 'cut' the network using MBs. Given a subnetwork S of a network N, the subnetwork consisting of the variables of N not included in S can be 'cut' into two parts: a subnetwork R and a Markov Boundary B (which is also a subnetwork) separating S and R. Note that the subnetwork R can be empty if the Markov Boundary of the subnetwork S includes all the variables of the network N not included in the subnetwork S.

Then, if the new network N is considered as consisting of R and B, and assuming the Markov Boundary B as the new subnetwork S, the same operation can be repeated, until the network cannot be cut anymore, i.e. when R becomes empty. For instance, applied to our example, if the variable A is considered as subnetwork S, then the subnetwork B is made up of the variables B, C, and D, and the subnetwork R of the variables F and G, as illustrated in Figure 3.

At this stage, if the starting subnetwork S and all the successive subnetworks B are gathered, a set of MBs containing all the variables of the network is obtained. Each of these boundaries, by definition, only depends on the ones directly adjacent to it, so it is not necessary to consider more than two consecutive MBs simultaneously to perform the probability calculations. In order to take into account the dependence with the two neighboring boundaries, the window is sliding, i.e. the boundaries n-1 and n are first considered, then n and n+1.

In the case of Viterbi inference, the objective is to propagate probabilistic information from one Markov Boundary to the next, in order to scan the entire network while controlling the complexity of the operation. To do this, the joint distribution of the starting subnetwork S must be completely known. In

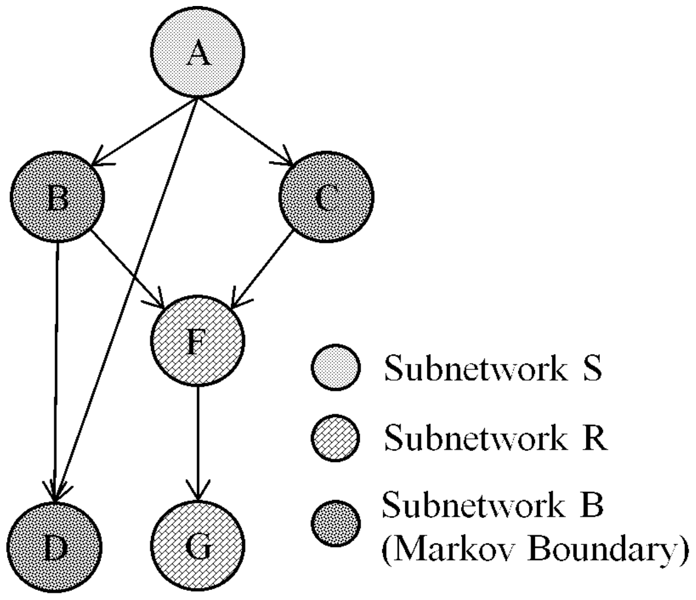


**Figure 3:** Example of "cutting" using the Markov Boundary.

practice, this translates into a starting subnetwork 'upstream' of the network, i.e. having no parent.

If the Markov Boundary determination procedure is applied to our example, the starting point would be variable A, the first Markov Boundary variables B, D, and C, the second F, and the third G, as shown in Figure 4.

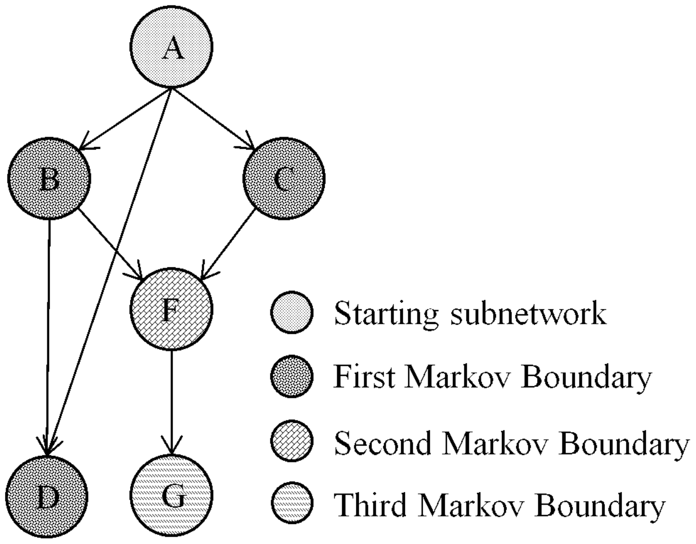


**Figure 4:** Cutting a network into a sequence of Markov Boundaries.

### Viterbi Algorithm

Concretely, the Viterbi inference algorithm is applied to the sequence of MBs, and proceeds in three steps:

The computation of the probability of each possible state of the starting subnetwork, given the observations (i.e. if a state is incompatible with the observations, its probability is null).

For each Markov Boundary and each possible state of this boundary, the registration of the state of the previous boundary giving the highest probability of the current boundary state considered, and calculation of this probability.

Recovering the states of the successive MBs giving the highest probability at the end of the network scan.  They constitute the most probable global state of the network. The Viterbi inference procedure is formalized in Algorithm 1.

$O_o^v$ represents the observation of variable $v$ in realisation $o$, *States*$(MB_r)$ denotes the set of possible states of Markov Boundary $r$, and $i_v$ represents the value of variable $v$ in state of Markov Boundary $i$. ${}^{j}MB_r^i$ denotes the state of variable $j$ in Markov Boundary $r$ in state $i$**.** $A_r^i$ is the element of the structure recording the antecedents associated to state $i$ of Markov Boundary $r$. $par(j) = ({}^{par(j)}MB_r^{\mathrm{i}}, {}^{par(j)}MB_{r-1}^k)$ means that the parents of $j$ can be both in the current Markov Boundary and in the previous one. $S_r$ denotes the rth element of $S$**.**

The complexity of this algorithm is $O(O \times MB \times N_s^{M \times N_v + N_v})$, where $O$ is the number of network realisations to infer, $MB$ is the number of Markov Boundaries, $N_v$ is the typical number of variables in a Markov Boundary, $N_s$ is the typical number of states of a variable, and $M$ is the typical missing observations ratio within a realisation.

## AMBV ALGORITHM

In this section, we describe in detail AMBV, the proposed algorithm to compare to Viterbi algorithm.

### Algorithm

AMBV automatically generates MBs for any BN while at the same time precalculating probabilities independent from observations, and saving them into structures later used during inference of the most probable state given the observations. This algorithm does not claim to find the best possible MBs, but only a good set of boundaries ensuring the inference success, in a simple and fast way.

The overall flow of the algorithm is as follows. The initial current Markov Boundary is formed by placing the variables with no parent in it.

As long as there are variables in the current Markov Boundary, the following procedure is repeated. The order of the variables in the set consisting of the current and previous Markov Boundary is optimized, so that certain subsets of states can be skipped when scanning all possible transitions. The probabilities of transitions from each state of the previous Markov Boundary (if it exists) to each state of the current Markov Boundary are computed, considering no observation, and each possible transition (probability greater than 0) is stored, as well as its associated probability. This allows reducing to the maximum the considered transitions from one Markov Boundary to the next. The states

```
Algorithm 1: Viterbi algorithm applied to BNs
  Data: network $N$, sequence $MB$ of the $R$ Markov Boundaries of the network, observation set $O$
  Result: Most probable state $S$ of the network and its probability $P_{max}$
1 for $o = 1$ to $Card(O)$ do
2     for $i = 1$ to $Card(States(MB_1))$ do
3         if $\nexists v \in MB_1 / O_o^v \neq i_v$ then
4             $P(MB_1^i) = \prod_{j \in MB_1} P(j = {}^jMB_1^i | par(j) = {}^{par(j)}MB_1^i)$;
5         else
6             $P(MB_1^i) = 0$;
7         end
8         $A_1^i = 0$;
9     end
10    for $r = 2$ to $R$ by steps of $1$ do
11        for $i = 1$ to $Card(States(MB_r))$ do
12            if $\nexists v \in MB_r / O_o^v \neq i_v$ then
13                $P(MB_r^i) = \max_{k \in MB_{r-1}} P(MB_{r-1}^k) \prod_{j \in MB_r} P(j = {}^jMB_r^i | par(j) = ({}^{par(j)}MB_r^i, {}^{par(j)}MB_{r-1}^k))$;
14                $A_r^i = \operatorname{argmax}_{j \in MB_r} P(j = {}^jMB_r^i | par(j) = ({}^{par(j)}MB_r^i, {}^{par(j)}MB_{r-1}^k))$;
15            else
16                $P(MB_r^i) = 0$;
17                $A_r^i = 0$;
18            end
19        end
20    end
21    $P_{max} = \max_{i \in MB_R} P(MB_R^i)$;
22    $S_R = \operatorname{argmax}_{i \in MB_R} P(MB_R^i)$;
23    for $r = R$ to $2$ by steps of $-1$ do
24        $S_{r-1} = A_r^{S_r}$;
25    end
26 end
```

**Algorithm 1:** Viterbi algorithm applied to BNs.

of the current Markov Boundary that can be accessed are also stored, in order to reduce the states considered for each Markov Boundary in subsequent iterations. Then, each variable in the current Markov Boundary votes for its children, and each child in turn votes for its own children. All variables that have at least one parent in the current Markov Boundary, have received at least one vote from each of their parents (and are not part of the current Markov Boundary), are put into the next Markov Boundary. If all the children of a variable in the current Markov Boundary are not included in the next Markov Boundary, the current Markov Boundary, or one of the previous MBs, that variable is put in the next Markov Boundary. Afterwards, the current Markov Boundary becomes the previous one and the next Markov Boundary becomes the current one. Finally, when there are no more variables in the current Markov Boundary, for each realization of the network in the dataset, the stored data is used to compute at a minimal cost the most probable path given the observations, using the principle of the Viterbi algorithm. The algorithm is formalized in Algorithms 2, 3 and 4.

JMB stands for Jointed Markov Boundaries, the junction of the variables of two consecutive MBs, and OJMB is it ordered version. We precise that *States* (*OJMB*) is swept through by systematically incrementing the variables of *OJMB* in ascending order. The states skipped inside the loop $s \in$ *States (OJMB)* count as having been swept through, and so contribute to fulfilling the end loop criterion.

```
Algorithm 2: AMBV - Initialization
  Data: network $N$, observations $O$
  Result: Most probable state $S$ of the network and its probability $P_{max}$
for $\{i \in N/par(X_i) = \emptyset\}$ do
    $MB_1 = MB_1 \cup X_i$;
end
```

**Algorithm 2:** AMBV-Initialization.

```
Algorithm 3: AMBV - Optimization structure generation
$i = 1$;
while $MB_i \neq \emptyset$ do
    $JMB = MB_i \cup \{v \in MB_{i-1}/v \notin MB_i\}$;
    for $\{j \in JMB/\nexists v \in JMB/j \in par(v)\}$ do
        $OJMB = OJMB \cup j$; (Order essential)
        $JMB = JMB \backslash j$;
        $j$ is reinitialized;
    end
    for $s \in States(OJMB)$ do
        $Interruption = 0$;
        if $i > 1$ and $s_{OJMB^{Card(OJMB)}_{Card(MB_i)+1}} \notin S_{i-1}$ then
            $Interruption = Card(MB_i) + 1$;
        else
            $P^s_{MB_i} = 1$;
            for $k = Card(MB_i)$ to 1 by steps of -1 do
                if $P(OJMB_k = s_{OJMB_k}|par(OJMB_k) = s_{par(OJMB_k)}) \neq 0$ then
                    $P^s_{MB_i} = P^s_{MB_i} P(OJMB_k = s_{OJMB_k}|par(OJMB_k) = s_{par(OJMB_k)})$;
                else
                    $Interruption = k$;
                    Interrupt $k$ loop;
                end
            end
        end
        if $Interruption = 0$ then
            ${}_{i-1}^{\;\;i}T^{s_{OJMB_1^{Card(MB_i)}}}_{s_{OJMB^{Card(OJMB)}_{Card(MB_i)+1}}} = P^s_{MB_i}$;
            Add $s_{OJMB_1^{Card(MB_i)}}$ to $S_i$
        else
            $s_{OJMB_{Interruption}} + +$;
            $s_{OJMB_1} = 0$;
        end
    end
    for $j \in MB_i$ do
        for $k \in chi(j)$ do
            $V(k, j) + +$;
            for $l \in chi(k)$ do
                $V(l, k) + +$;
            end
        end
    end
    for $\{j \in N \backslash \bigcup_{m=1}^{i} MB_m / \exists u \in par(j)/u \in MB_i$ and $\forall v \in par(j), V(j, v) > 0\}$ do
        $MB_{i+1} = MB_{i+1} \cup j$;
    end
    for $\{j \in MB_i/\exists v \in chi(j)/v \notin \bigcup_{m=1}^{i+1} MB_m\}$ do
        $MB_{i+1} = MB_{i+1} \cup j$;
    end
    $i + +$;
end
```

**Algorithm 3:** AMBV-Optimization structure generation.

$X_i$ represents the ith network variable, *par(v)* represents all of *v* parents, and $s_k$ denotes the state of variable *k* in states of Markov Boundary. *S* is the structure containing the accessible states of each Markov Boundary. So, $S_{i-1}$ contains all the accessible states of Markov Boundary *i*-1.

```
Algorithm 4: AMBV - Inference
for $o = 1$ to $Card(O)$ do
    for $i = 1$ to $Card(MB)$ by steps of 1 do
        for $t \in T^i_{i-1}$ do
            if $\nexists v \in MB_i/O^v_o \neq t(target)_v$ then
                $P(MB_i^{t(target)}) = \max_{k \in {}^{t(target)}T^i_{i-1}} P(MB^k_{i-1})\, {}^{t(target)}_{\quad\;\; k}T^i_{i-1}$;
                $A_i^{t(target)} = \operatorname{argmax}_{k \in {}^{t(target)}T^i_{i-1}} P(MB^k_{i-1})\, {}^{t(target)}_{\quad\;\; k}T^i_{i-1}$;
            else
                $P(MB_i^{t(target)}) = 0$;
                $A_i^{t(target)} = 0$;
            end
        end
    end
    $P_{max} = \max_{i \in MB_{Card(MB)}} P(MB^i_{Card(MB)})$;
    $S_{Card(MB)} = \operatorname{argmax}_{i \in MB_{Card(MB)}} P(MB^i_{Card(MB)})$;
    for $i = Card(MB)$ to 2 by steps of $-1$ do
        $S_{i-1} = A_i^{S_i}$;
    end
end
```

**Algorithm 4:** AMBV-Inference.

$P^s_{MB_i}$ is a variable gradually calculated, representing the probability of Markov Boundary *i* given considered states of *OJMB*. *Card (MB$_i$)* represents the number of variables in *MB$_i$*, *OJMB$_k$* represents the kth variable of the previous and current Markov Boundary ordered, and $s_{OJMB_1^{Card(MB_i)}}$ denotes the states in *s* of the variables in *OJMB* from 1 to *Card(MB$_i$)*.

*T* is a three-dimensional structure containing the transition data, whose first dimension is characterized by the two Markov Boundaries concerned by the transition, the second dimension is characterized by the target state of the transition, and the third dimension by the source state. Thus, ${}_{i-1}^{\;\;i}T^{s_{OJMB_1^{Card(MB_i)}}}_{s_{OJMB^{Card(OJMB)}_{Card(MB_i)+1}}}$ is the element of structure *T* giving the probability of the transition from state $s_{OJMB^{Card(OJMB)}_{Card(MB_i)+1}}$ of Markov Boundary *i*-1 to state $s_{OJMB_1^{Card(MB_i)}}$ of Markov Boundary *i*. *chi(j)* represents the children of variable *j*, and *V(k,j)* represents the number of votes for variable *k* from variable *j*.

*Card(MB)* is the number of network Markov Boundaries, $O^v_o$ denotes the observation of variable *v*, *t(target)* is the target state of the transition *t*, and *t(target)$_v$* denotes the state of variable *v* in *t(target)*.

The complexity of this algorithm is $O(MB \times ((1-S)N_s)^{2N_v}) + o(O \times MB \times ((1-S)N_s)^{M \times N_v + N_v})$, where *S* is the sparsity of the conditional probability tables of the variables.

## Example of Markov Boundaries Generation

Part of the presented algorithm will now be applied to a network example, in order to show how MBs are generated. Figure 5 graphically illustrates the main elements of the following description of the mechanism. Each type of dashed outline

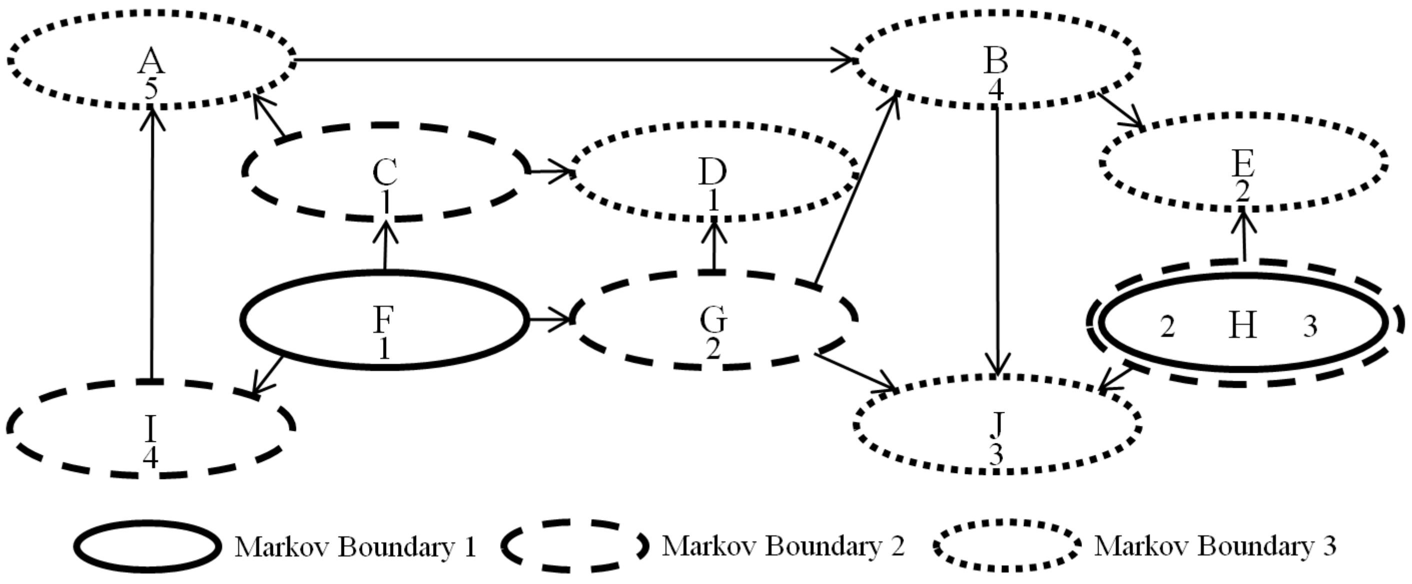


**Figure 5:** Markov Boundaries generated on a network example.

represents a Markov Boundary, and the numbers indicate the position of the variable in the Markov Boundary concerned.

First, the variables F and H are placed in Markov Boundary 1, because they are the only ones without parents.

Then, adding the variables from the previous boundary is a non-operation because the boundary does not exist. Also, since the variables have no children in boundary 1, they are not reordered.

Variable F votes for its children C, G, and I, then C votes for its own children A and D, G votes for B, D and J, and I votes for A. H votes for E and J, which do not vote because they have no children.

C, as well as G and I received a vote from their unique parent, which is in Markov Boundary 1, so they are included in Markov Boundary 2. A and D did receive votes from all their parents, but none of them belong to Markov Boundary 1, so they are not included in Markov Boundary 2. Variables B, J, and E did not receive a vote from each of their parents, so they are not included in Markov Boundary 2 either.

Since the variables E and J are not included in MBs 1 or 2, the variable H must also be inserted in Markov Boundary 2.

Markov Boundary 2 therefore contains the variables C, G, H, and I.

None of these variables has a child in Markov Boundary 2, so they are not reordered.

Variable C votes for its children A and D, then A votes for B, and D does not vote because it has no children. G votes for B, D and J, then B votes for E and J, and D and J do not vote because they have no children. H votes for E and J, which do not vote because they have no children. I votes for A, which votes for B.

A, D, B, J, and E have received one or more votes from each of their parents, at least one of whom being in Markov Boundary 2, so they are included in Markov Boundary 3.

The Markov Boundary 3 thus contains the variables A, B, D, E and J.

Then, a systematic scan of all variables in Markov Boundary 3 is performed. Variable A having one child in Markov Boundary 3, it is put aside. B having two children in Markov Boundary 3, it is set aside. Since D has no children in Markov Boundary 3, it is placed in first position of the ordered Markov Boundary 3 and removed from the original Markov Boundary 3. Variable A is then reconsidered; it still has one child in the original Markov Boundary 3, so it is set aside again. B still has two children in the original Markov Boundary 3, so it is set aside. E has no children in the original Markov Boundary 3, so it is placed in second position of the ordered Markov Boundary 3, and removed from the original Markov Boundary 3. Once again, A is considered; it still has one child in the original Markov Boundary 3, so it is set aside once more. B now has one child in the original Markov Boundary 3, so it is still set aside. J has no children in the original Markov Boundary 3, so it is added in third place to the ordered Markov Boundary 3. Back to A: it still has one child in the original Markov Boundary 3, so it is set aside again. B now has no more children, so it is placed in fourth position of the ordered Markov Boundary 3. Finally, since A is now alone in the boundary, it has no more children, so it is added in last position of the ordered Markov Boundary 3. The Markov Boundary 3, after being ordered, now contains, in order, D, E, J, B, and A.

Since none of the variables in Markov Boundary 3 have children outside it, no variable can be added to Markov Boundary 4, which is thus empty, ending the generation of MBs.

## EXPERIMENTAL COMPARISON

In this section we compare experimentally the complexity of AMBV to the Viterbi algorithm we presented.

### Simulation Context

There are two cases of comparison: when the network is a DBN, and when it is a BN. In the first case, the MBs used by the Viterbi algorithm used as a reference are directly deduced from the time slices. In the second case, a set of MBs is generated using the mechanism employed in AMBV, and given to the reference Viterbi algorithm. In all cases, AMBV computes its own MBs.

The two algorithms will be compared in terms of computational time for the same inference task and on the same platform (2 Intel Xeon CPU E5-2620 v4 @ 2.1GHz and 64GB of RAM), excluding the generation time of the MBs by the reference Viterbi algorithm.

The six parameters of the inference whose influence is studied are gathered in Table 1. First, the Number of Variables per (Time) Slice, set to 3 by default, and evaluated on the interval [1;5] by steps of 2, in campaign 1. It can be noted that only one variable corresponds to a simple Markov Model. Then, the Number of States per Variable, fixed at 4 by default, and evaluated on the interval [2;6] by steps of 2, in campaign 2. Next, the Sparsity ratio, set to 0 by default, and evaluated on the interval [0;0.8] by steps of 0.4, in campaign 3. Then, the Number of (Time) Slices, set to 10 by default, and evaluated for the values 3, 10, and 30, in campaign 4. Next, the Missing Values Percentage, set to 20 by default, and evaluated on the interval [20;100] by steps of 40, in campaign 5. Finally, the Number of Realizations to be evaluated, i.e., inferences to perform, which is evaluated on the interval [1;1000] by logarithmic step, in all campaigns. Only one parent per variable is considered, as it has nearly no influence on computation time. In order to smooth the results, an averaging is performed over ten repetitions of each simulation. Note that some curves may present artefacts when the number of realizations involved is low (<10).

The graph in Figure 6 shows the computation time of both Viterbi and AMBV algorithms, as a function of the number of realizations, and parameterized by the number of variables per slice.

It can be seen that Viterbi algorithm computation time is proportional to the number of inferences performed, which is coherent with its complexity. On the other hand, AMBV computation time also displays a linear growth with the number of realizations; however, its coefficient decreases when the number of variables per slice increases. This behaviour is perfectly coherent with the complexity of AMBV, and is explained by the fact that the optimization structure generation's complexity increases far quicker than the inferences. Both algorithms also follow an exponential law whose exponent is the number of variables per slice, as expected from their complexities.

This graph tells us that above a certain number of realizations, AMBV is always faster than Viterbi, and that this threshold increases with the number of variables per slice. Moreover, the size of the number of realizations gaps in-between the consecutive thresholds slowly decrease as the number of variables per slice increases.

The graph in Figure 7 displays the computation time of both AMBV and Viterbi algorithms as a function of the number of realizations, and parameterized by the number of states per variable. The two algorithms follow an exponential law whose base is the number of states per variable. As observed in most graphs, AMBV always ends up better than Viterbi, and the higher the number of states per variable, the smaller the realization gap between the intersections of two consecutive couples of curves.

The graph in Figure 8 displays the computation time of both AMBV and Viterbi algorithms as a function of the number of realizations, and parameterized by the sparsity of the distributions. It can be observed that sparsity has no effect on Viterbi algorithm, as we might expect from its complexity. On the other hand, AMBV follows an exponential law whose base is the complement to 1 of the sparsity (1-S), which means its computation time greatly

**Table 1:** Campaigns parameters.

| Parameter | Campaign 1 | Campaign 2 | Campaign 3 | Campaign 4 | Campaign 5 |
|---|---|---|---|---|---|
| Number of Variables per Slice | 1;3;5 | 3 | 3 | 3 | 3 |
| Number of States per Variable | 4 | 2;4;6 | 4 | 4 | 4 |
| Sparsity | 0 | 0 | 0;04;0;8 | 0 | 0 |
| Number of Slices | 10 | 10 | 10 | 3;10;30 | 10 |
| Missing Values Percentage | 20 | 20 | 20 | 20 | 20;60;100 |
| Number of Realizations | 1 to 1000 by log 10 steps | 1 to 1000 by log 10 steps | 1 to 1000 by log 10 steps | 1 to 1000 by log 10 steps | 1 to 1000 by log 10 steps |
| Number of Parents per Variable | 1 | 1 | 1 | 1 | 1 |
| Number of Repetitions | 10 | 10 | 10 | 10 | 10 |

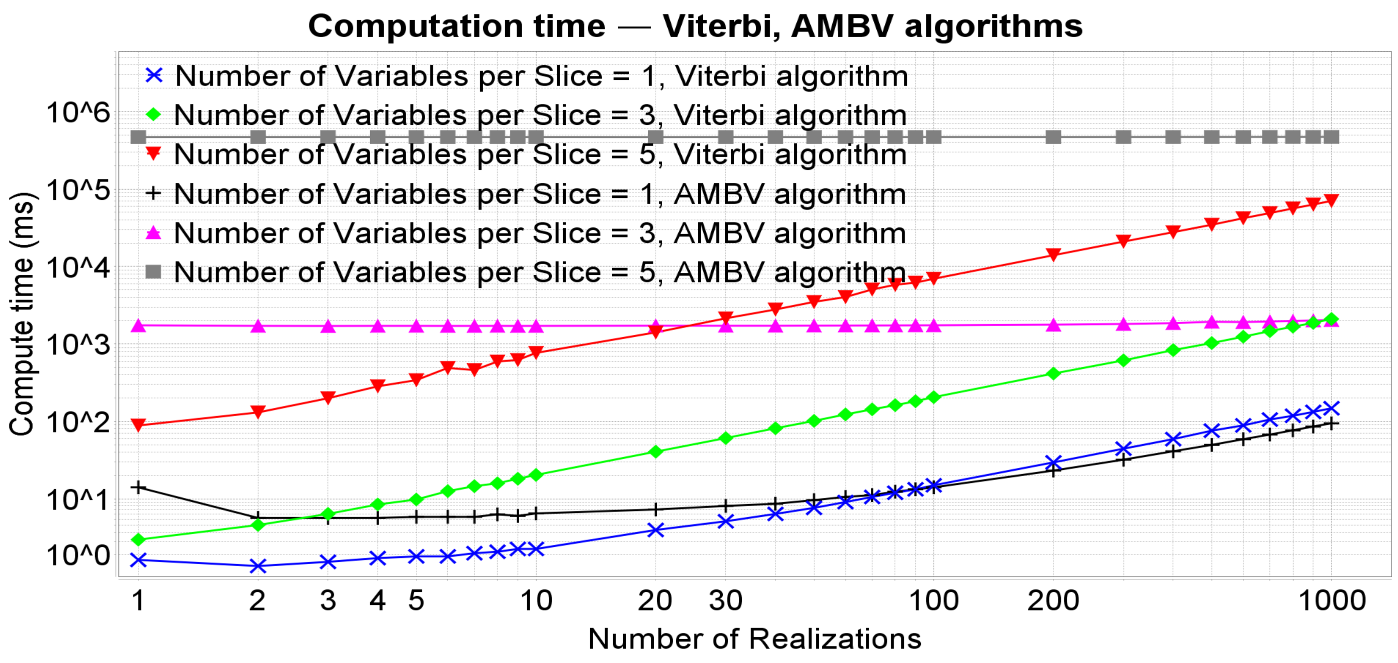


**Figure 6:** Computation time of Viterbi and AMBV algorithms as a function of the number of realizations and parameterized by the number of variables per slice.

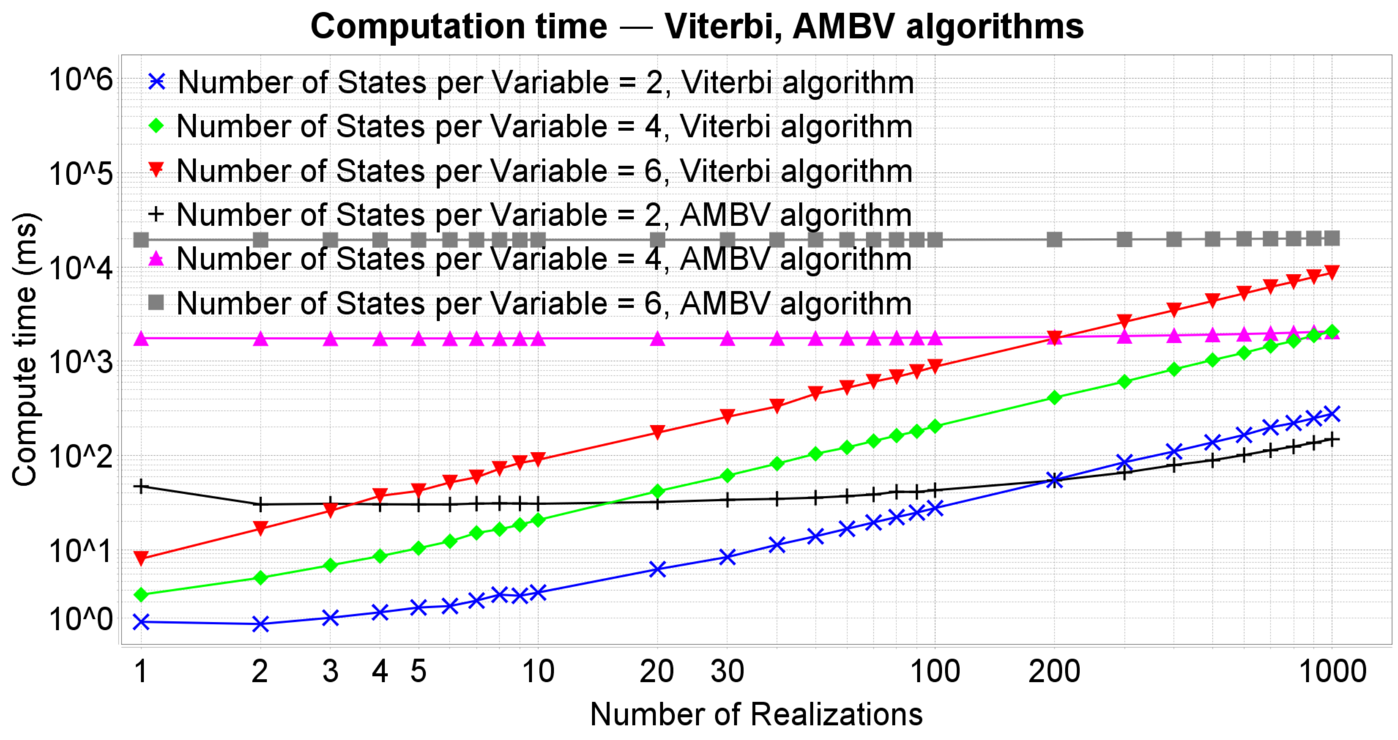


**Figure 7:** Computation time of Viterbi and AMBV algorithms as a function of the number of realizations and parameterized by the number of states per variable.

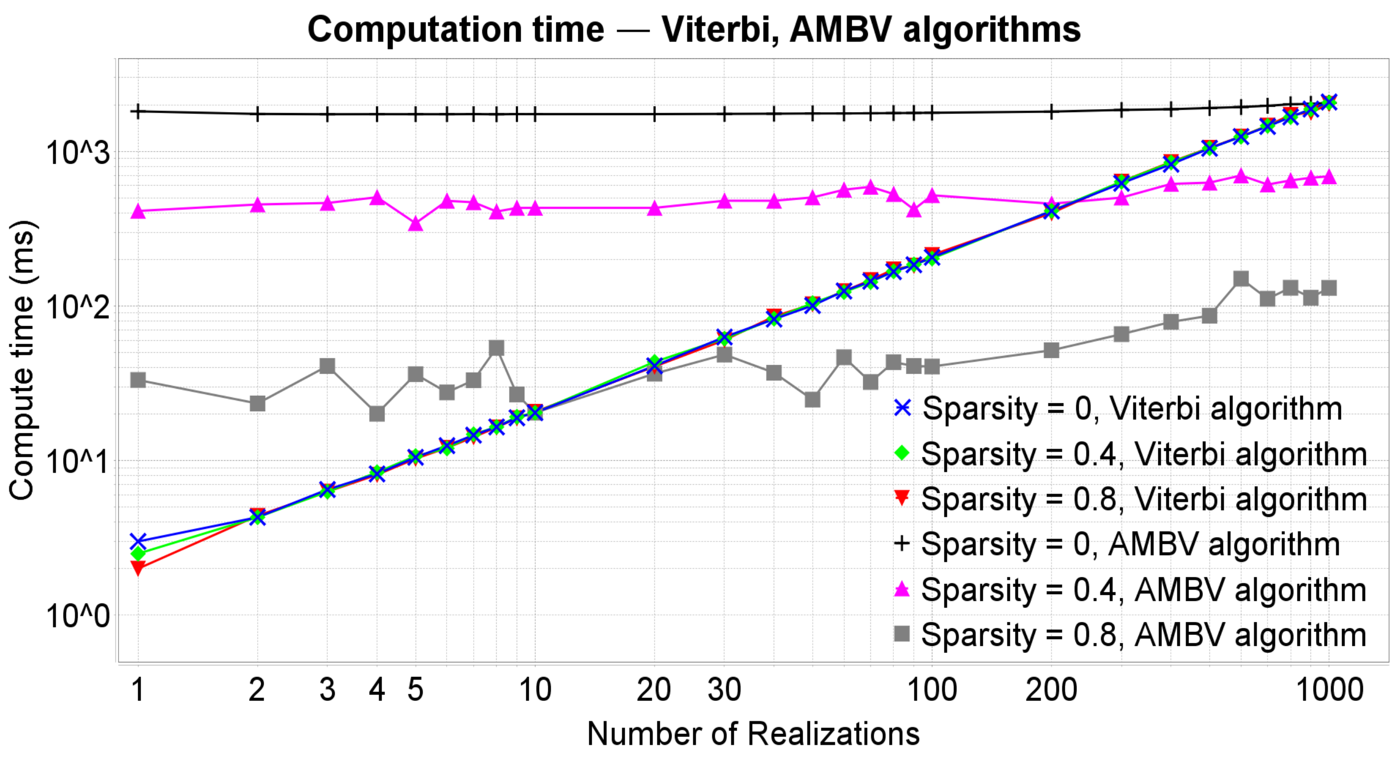


**Figure 8:** Computation time of Viterbi and AMBV algorithms as a function of the number of realizations and parameterized by the sparsity.

decreases when the distributions become sparse. It can be noted that the effect is more noticeable on the optimization structure generation part than the inference part. The graph clearly shows us that when sparsity increases, AMBV algorithm becomes better than Viterbi algorithm with far less inferences needed.

The graph in Figure 9 displays the computation time of both AMBV and Viterbi algorithms as a function of the number of realizations, and parameterized by the number of slices. Both algorithms computation times increase linearly with the number of slices, in coherence with their complexities. However it is not obvious for AMBV for less than 10 slices, as the temporal slices are not directly used as MBs, like in the case of Viterbi algorithm. So, AMBV might use a slightly different number of boundaries, but the difference may be huge when the total number of slices is small. It can also be seen that, as the number of slice increases, AMBV needs more and more realizations to be more efficient than Viterbi, but it always ends up better. The number of realizations gaps between the intersections of each couple of curves also decreases as the number of slices increases.

Finally, the graph in Figure 10 shows the computation time of both AMBV and Viterbi algorithms as a function of the number of realizations, and parameterized by the missing values percentage. The Viterbi algorithm behaves as expected from the complexity, i.e., it follows an exponential law whose exponent is the missing values percentage. AMBV was expected to display the same behaviour, but the missing values percentage ends up with no influence on its computation time. Two reasons explain this: first and foremost, the inference part of the computation is very short compared to the optimization structure generation (less than 1%), and this is true whatever the parameters. The second reason is that the probability computation performed in each of the inference threads is far shorter than the time to create and destroy the thread, in the case of low parameter values such as the ones used here. The two phenomena add up, leading to a total computation time variance being far more important than the computation time saved with less missing values. However, with a more optimized implementation, it would be possible to take advantage of less missing values.

## CONCLUSION

In this paper, we presented the Automatic Markov Boundaries construction optimized Viterbi (AMBV) algorithm, which performs Viterbi inferences in any discrete Bayesian Network by generating its own Markov Boundaries. It takes advantage of the fact that when doing multiple inferences with the same network, everything independent from the observations, i.e., the transition probabilities from one Markov Boundary to the following one, can be pre-calculated to save some computation power. It also cleverly orders the variables within each Markov Boundary to enable skipping some boundary state subsets when the distributions are sparse. We compared its computation time to that of a classic Viterbi for the same task and on the same platform. AMBV ended up being always faster when enough

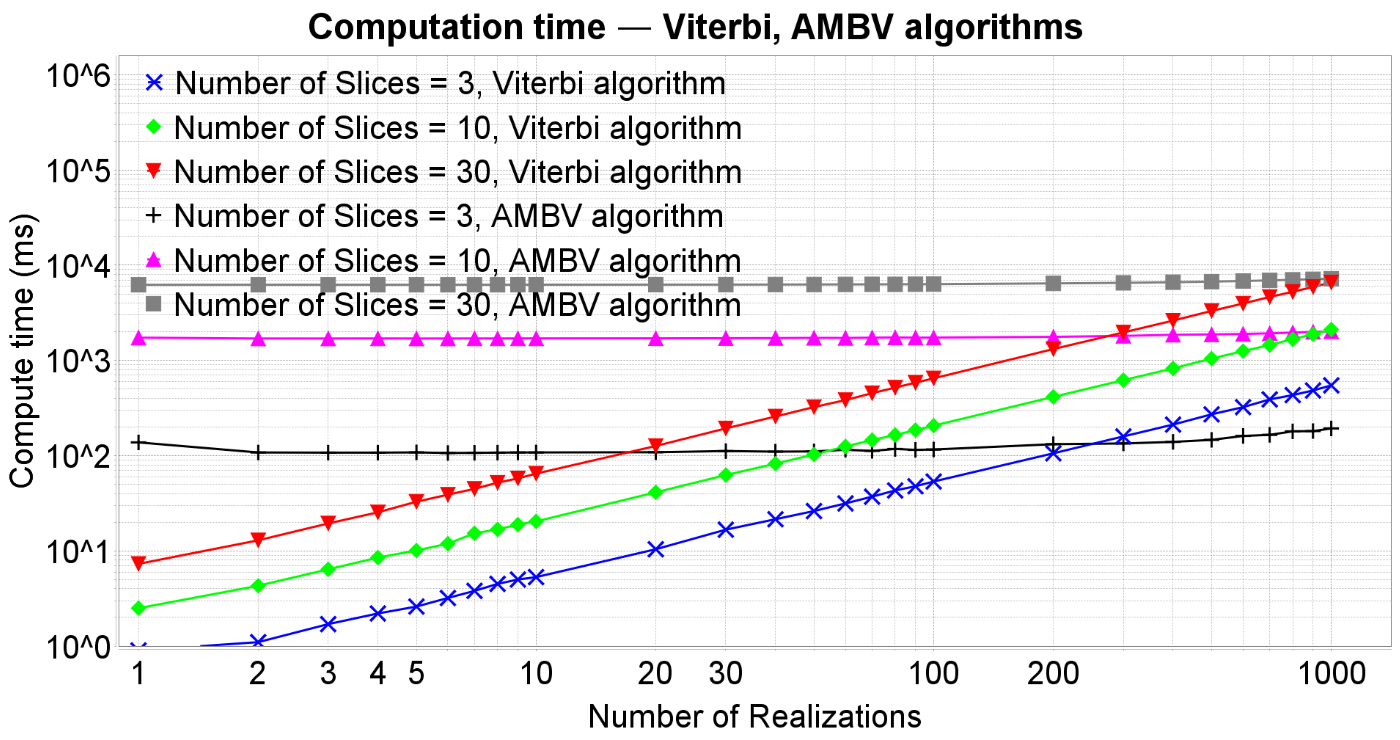


**Figure 9:** Computation time of Viterbi and AMBV algorithms as a function of the number of realizations and parameterized by the number of slices.

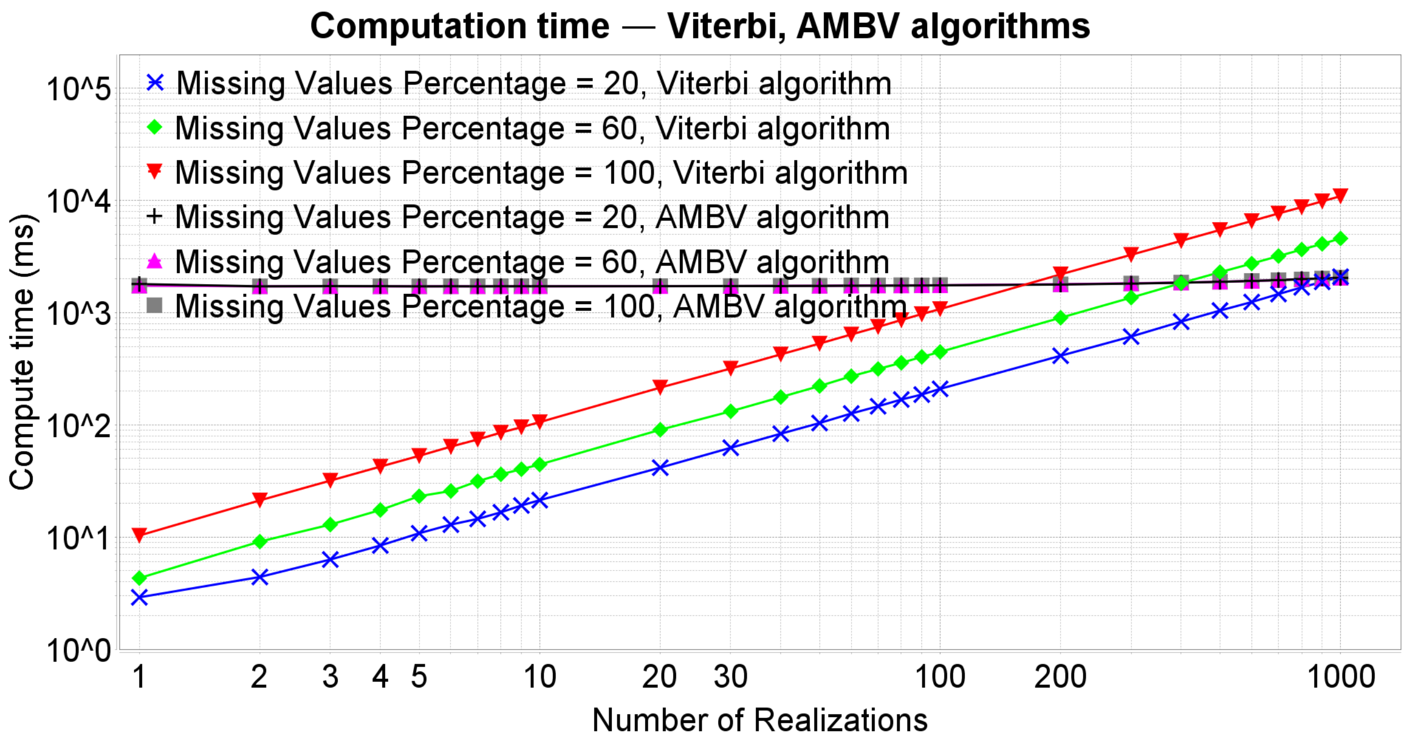


**Figure 10:** Computation time of Viterbi and AMBV algorithms as a function of the number of realizations and parameterized by the missing values percentage.

realizations are performed; this threshold generally increases with the complexity of the network and decreases with the sparsity of the distributions. However, the gaps in terms of realizations between two consecutive thresholds also decrease with the complexity of the network, so it becomes more efficient as the complexity increases. We can conclude that AMBV is relevant for large networks, but underperforms on small networks because of its important overhead. To further increase AMBV performance, the optimization structure generation could be parallelized, like the inference part and Viterbi are. Moreover, when switching from one Markov Boundary state to the next, only one variable changes state, so it may be interesting to only recalculate the necessary probabilities. Finally, it might be useful to design a heuristic process to choose the variables in each Markov Boundary, in order to limit the number of each boundary states, with the help of a first sweeping across the network, for instance.

## AUTHORS' CONTRIBUTION

Pierre-Samuel Gréau-Hamard: conceptualization, literature search, data collection, algorithm formulation, writing, result analysis and editing. Moïse Djoko-Kouam: reviews paper content and finalization. Yves Louët: : organization and analysis.

## CONFLICT OF INTEREST

The authors declare that there is no conflict of interest.

## REFERENCES

1. Han M, Liu X, Zhang W, Wang M, Bu W, Chang C, *et al*. TSMiner: A novel framework for generating time-specific gene regulatory networks from time-series expression profiles. Nucleic Acids Research. 2021;49(18):e108. DOI: 10.1093/nar/gkab629.
2. Irsalinda N, Haswat H, Sugiyarto S, Fitrianawati M. Hidden markov model for sentiment analysis using viterbi algorithm. EKSAKTA. 2021;2(1):18-23.
3. Bai L, Zhang Z, Hu J. Voice activity detection based on deep neural networks and Viterbi. IOP Conference Series: Materials Science and Engineering. 2017;231(1):12-42. DOI: 10.1088/1757-899x/231/1/012042.
4. Figueroa-Angulo JI, Savage J, Bribiesca E, Escalante B, Sucar LE. Compound Hidden Markov Model for Activity Labelling. International Journal of Intelligence Science. 2015;5(5):177-95.
5. Song Y, Adobah B, Qu J, Liu C. Segmentation of Ordinary Images and Medical Images with an Adaptive Hidden Markov Model and Viterbi Algorithm. Current Signal Transduction Therapy. 2020;15(2):109-23.
6. Mouafo SRT, Vaton S, Courant JL, Gosselin S. A tutorial on the EM algorithm for Bayesian networks: Application to self-diagnosis of GPON-FTTH Networks. In: Proceedings of IWCMC 2016: 12th International Wireless Communications and Mobile Computing Conference, Paphos, Cyprus. 2016:369-76.
7. Murphy K. Dynamic bayesian networks: Representation, inference and learning. Ph.D. dissertation. Fall. 2002.
8. Rabiner L, Juang B. An introduction to hidden Markov models. IEEE ASSP Magazine. 1986;3(1):4-16.
9. Forney GD. The viterbi algorithm. In Proceedings of the IEEE. 1973;61(3):268-78.
10. Mihajlovic V, Petkovic M. Petkovic, Dynamic Bayesian Networks: A State of the Art. CTIT Technical Report Series. 2001;34(1):1-37.

**Cite this article:** Gréau-Hamard PS, Djoko-Kouam M, Louët Y. AMBV: An Optimized Generic Viterbi Algorithm for Bayesian Networks. J Scientometric Res. 2024;13(3):745-56.