Fast Algorithms for Robust Classification with Bayesian Nets

Alessandro Antonucci [*] and Marco Zaffalon

IDSIA, Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, Galleria 2, CH-6928 Manno (Lugano), Switzerland

Abstract

We focus on a well-known classification task with expert systems based on *Bayesian networks*: predicting the state of a target variable given an incomplete observation of the other variables in the network, i.e., an observation of a subset of all the possible variables. To provide conclusions robust to near-ignorance about the process that prevents some of the variables from being observed, it has recently been derived a new rule, called *conservative updating*. With this paper we address the problem to efficiently compute the conservative updating rule for robust classification with Bayesian networks. We show first that the general problem is *NP-hard*, thus establishing a fundamental limit to the possibility to do robust classification efficiently. Then we define a wide subclass of Bayesian networks that does admit efficient computation. We show this by developing a new classification algorithm for such a class, which extends substantially the limits of efficient computation with respect to the previously existing algorithm. The algorithm is formulated as a *variable elimination* procedure, whose computation time is linear in the input size.

Key words: Bayesian networks, missing data, conservative updating rule, credal classification, valuation algebras, imprecise probabilities.

1 Introduction

Probabilistic expert systems yield conclusions on the basis of *evidence* about a domain. For example, systems based on *Bayesian networks* [1] (see Section 2.1) are queried for updating the confidence on a target variable given

[⋆] This research was partially supported by the Swiss NSF grant 200020-109295/1.

[*] Corresponding author

Email addresses: alessandro@idsia.ch (Alessandro Antonucci), zaffalon@idsia.ch (Marco Zaffalon).

an evidence, i.e., after observing the value of other variables in the network model. Very often, at the time of a query, only a subset of all the variables is in a known state, as there is a so-called *missingness process* that prevents some variables from being observed. This is a crucial point. The traditional way to update beliefs in probabilistic expert systems relies on Kolmogorov's conditioning rule. In order to yield correct conclusions, such a rule needs that the missingness process is explicitly modelled, or at least that it does not act in a selective way (i.e., that it is not malicious in producing the missingness). Unfortunately, the missingness process may be difficult to model, and assuming that it is unselective is equivalent to assuming the well-known *missing at random* (MAR) condition [2], which is often unrealistic [3].

To address such a fundamental issue, de Cooman and Zaffalon [4] have recently derived a new rule to update probabilities with expert systems in the case of near-ignorance about the missingness process. As a more realistic model of this condition of partial information, the new, so-called, *conservative updating rule* (or CUR), yields lower and upper probabilities in general, as well as partially determined decisions. With classification problems, for instance, where the goal is to predict the state of the target variable (also called *class variable*) given an evidence, CUR leads to set-based classifications, or, in other words, to *credal classifiers* [5] (see Section 2.2). De Cooman and Zaffalon have indeed specialized CUR to solve classification problems with Bayesian networks. Yet, their algorithm is efficient only on a relatively limited class of Bayesian networks: those in which the *Markov blanket*[1] of the class variable together with the variable itself forms a *polytree* (also called a *singly connected graph*), that is, a graph that becomes a tree after dropping the orientation of the arcs. Two natural questions arise in relationship with the above algorithm: is it possible to provide an algorithm for CUR-based classification that is similarly efficient on more general network structures? And, at a more fundamental level, what are the limits of efficient computation posed by the nature of the problem?

With this paper we address both questions. Initially, we prove the hardness of the problem, thus solving the second question: doing classification with CUR on Bayesian nets is shown to be *NP-hard* in Section 3. This parallels analogous results obtained for Bayesian nets that implement the traditional updating [6]; in those cases, the algorithms are efficient when the entire graph is a polytree, and are exponential with more general, so called, *multiply connected graphs.*

Then we address the first question by developing a new algorithm that substantially extends the limits of efficient computation with respect to de Cooman and Zaffalon's original algorithm. We achieve this goal, which is relatively involved from the technical point of view, in different steps. We firstly introduce

[1] The set of nodes made by the parents, the children, and the parents of the children of a given variable.

in Section 4.1 a new kind of network model, called *s-network*, that abstracts the main features of a CUR-based classification on a Bayesian net. Secondly, in Section 4.2, we show that our classification problem can be solved through suitable calculations on a corresponding s-network; an algorithm that implements this mapping is also provided. In Section 5.1 this particular calculation over s-networks is proved to be equivalent to a *variable elimination* procedure in the abstract framework of a *valuation algebra* [7] (see Sect. 2.3). In Section 5.2, a strategy that defines a particular order in which the variables should be eliminated is provided for the special case of classification problems such that the corresponding s-network is a polytree (or a collection of them, i.e., a *polyforest*). In this way it is possible to provide a linear time algorithm performing these calculations (Section 5.3). That concerns also many cases when the class variable with its Markov blanket forms a multiply connected graph in the original Bayesian net. This, together with the fact that the complexity of CUR-based classification depends on the structure of the Markov blanket rather than that of the entire net, makes the new algorithm efficient on a truly large subset of Bayesian networks.

Overall, we develop a computational basis to do classification in expert systems when there is little knowledge about the process producing the missingness. This enables efficient computation to take place on a large subset of Bayesian networks, which is of course important for applications. General remarks about CUR-based classification are in Section 6, while comments on the results conclude the paper in Section 7. Some proofs of the Theorems and preliminary Lemmas are reported in Appendix A.

2 Setup

2.1 Bayesian Networks

Consider the random variables $A_0, \ldots, A_n$. Variable A_k $(k = 0, \ldots, n)$ takes generic value a_k from the finite set $\mathscr{A}_k$. The available information about the relationship between the random variables is specified by a (prior) mass function $p(A_0, \ldots, A_n)$, which we assume to be positive in the following.

The mass function $p(A_0, \ldots, A_n)$ can be conveniently provided by a domain expert using a Bayesian network. A *Bayesian network* is a pair composed of a directed acyclic graph and a collection of conditional mass functions. There is one-to-one correspondence between the nodes of the graph and the random variables $A_0, \ldots, A_n$. Accordingly, the same symbol is used to denote A_k and the related node; and 'node' and 'variable' are used interchangeably. Each node A_k holds a conditional mass function $p(A_k|\pi_{A_k})$ for each joint state π_{A_k}

of its direct predecessor nodes (or *parents*) Π_{A_k}.

Bayesian nets satisfy the *Markov condition*: every variable is independent of its nondescendant non-parents given its parents. From the Markov condition, it follows [1] that the joint probability $p(a_0,\ldots,a_n)$ is given by $p(a_0,\ldots,a_n) = \prod_{k=0}^{n} p(a_k|\pi_{A_k})$ for all the $(n+1)$-tuples $(a_0,\ldots,a_n) \in \times_{k=0}^{n} \mathscr{A}_k$, where π_{A_k} is the assignment to the parents of A_k consistent with $(a_0,\ldots,a_n)$.

For the purposes of the paper, we arbitrary choose A_0 as target node, aiming at predicting its state given values of some other nodes. In the following A_0 will be called *class variable* and will also be denoted by C, with generic value c from the set of classes $\mathscr{C} := \mathscr{A}_0$. The remaining variables will be called *attribute variables*, and their values *attributes*. We refer to this predictive problem as *classification*.

2.2 Robust Classification

In classification problems, we typically observe (or measure) only a subset of the attribute variables at the time of a query. In order to update probabilities about the class variable given the observations, there is a frequent habit to neglect the missing attribute variables after the conditioning bar. However, this method is justified only when the process responsible for the missingness is unselective, that is, when it creates the missingness without any specific purpose. More technically, this happens when the probability that a measurement is missing is the same irrespectively of the specific measurement. In this case we say that the process is MAR [2]. Unfortunately, MAR is quite a strong assumption [3] and for this reason MAR-based approaches are somewhat criticized (see also [8]).

Following a deliberately conservative approach, de Cooman and Zaffalon [4] have instead used *coherent lower previsions* [9], which are equivalent to closed convex sets of mass functions, to model the case of near-ignorance about the missingness process. This has led to a new rule to update beliefs in expert systems that is called conservative updating rule. In order to denote incomplete observations of the attribute variables (the class variable is clearly unobserved, as it is the variable to predict), let us use E for the subset of the attribute variables that are observed and e for their joint value. Let us denote by R the remaining attribute variables, whose values are missing. We also denote the set of their possible joint values by $\mathscr{R}$, and a generic element of that set by r. Observe that for every $r \in \mathscr{R}$, the attributes vector (e,r) is a possible *completion* of the incomplete observation $(E,R) = (e,*)$, where the symbol $*$ denotes missing values. The updated probability of the class variable given $(e,*)$ is an interval, according to the conservative updating rule, whose extremes are the following:

$$\underline{p}(c|e,*) := \min_{r\in\mathscr{R}} p(c|e,r) \tag{1}$$
$$\overline{p}(c|e,*) := \max_{r\in\mathscr{R}} p(c|e,r). \tag{2}$$

In this paper we are concerned with predicting the value of the class variable given $(e,*)$. This is equivalent to producing the set of the *undominated* classes according to the conservative updating rule. Say that class c' *credal-dominates*, or simply *dominates*, class c'', and write $c' \succ c''$, if $p(c'|e,r) > p(c''|e,r)$ for all $r \in \mathscr{R}$. A class is said to be undominated if there is no class that dominates it. This dominance criterion is a special case of *strict preference* proposed by Walley [9, Section 3.7.7]. In other words, the conservative updating rule generally produces set-based classifications, where each class in the output set should be regarded as a candidate *optimal* class. Classifiers that produce set-based classifications are also called *credal classifiers* by Zaffalon [5].

It is easy to show that testing whether $c' \succ c''$ can be carried out in the following equivalent way:

$$\min_{r\in\mathscr{R}} \frac{p(c',e,r)}{p(c'',e,r)} > 1. \tag{3}$$

Let us use π' and π'' to denote values of parent variables consistent with the completions (c',e,r) and (c'',e,r), respectively. Regarding C, let π denote the value of its parents consistent with (e,r). Furthermore, without loss of generality, let A_1, ..., A_m, $m \leq n$, be the *children* (i.e., the direct successor nodes) of C. Denote by B^+ the union of C with its Markov blanket. De Cooman and Zaffalon [4] show that the minimum in (3) can be computed by restricting the attention to B^+, in the following way:

$$\min_{\substack{a_j\in\mathscr{A}_j,\\ A_j\in B^+\cap R}} \left[\frac{p(c'|\pi_C)}{p(c''|\pi_C)} \prod_{i=1}^{m} \frac{p(a_i|\pi'_{A_i})}{p(a_i|\pi''_{A_i})} \right]. \tag{4}$$

Note that Expression (4) does not change by removing the arcs such that their second endpoint [2] is neither C nor one of its children. In the following, we will refer to B^+ just as the subgraph deprived of those negligible arcs.

2.3 Local Computations on Valuation Algebras

Many different formalisms for managing uncertainty in expert systems share a common algebraic structure based on elementary operations such as *aggrega-*

[2] Two nodes connected by an arc are said to be its *endpoints*. The first endpoint is the node from which the arc departs, while the second is the remaining node.

tion of knowledge and *focus* on part of the overall information. In this section we present a formal definition of this structure together with an algorithm for solving many computational tasks on this framework.

Let $\mathscr{V}$ be a finite collection of random variables over finite domains and Φ a set of abstract objects called *valuations*. Three operations are assumed to be defined over Φ and $\mathscr{V}$, namely a *labelling* $d : \Phi \rightarrow 2^{\mathscr{V}}$, a *combination* $\otimes : \Phi \times \Phi \rightarrow \Phi$ and a *variable elimination* $\mathrm{El} : \Phi \times \mathscr{V} \rightarrow \Phi$.

Every valuation $\phi \in \Phi$ is interpreted as a piece of knowledge about the possible values of the variables in $d(\phi) \subseteq \mathscr{V}$ and $d(\phi)$ is called *domain* of ϕ. Given two valuations $\phi, \psi \in \Phi$, the *combined valuation* $\phi \otimes \psi$ represents the aggregate knowledge coming from both ϕ and ψ. Finally, given a valuation $\phi \in \Phi$ and a variable $A \in \mathscr{V}$, $\mathrm{El}(\phi, A)$ represents a valuation that focuses on the knowledge associated to ϕ with no attention to what is related to A. We use also the notation ϕ^{-A} to denote $\mathrm{El}(\phi, A)$.

The 5-tuple $(\Phi, \mathscr{V}, d, \otimes, \mathrm{El})$ is said to be a *valuation algebra* (VA) [7] if the operations of labelling d, combination $\otimes$, and variable elimination El over the set of valuations Φ and the set of random variables $\mathscr{V}$, satisfy the following system of axioms:

(A1) Φ is commutative and associative under $\otimes$
(A2) If ϕ, $\psi \in \Phi$, then $d(\phi \otimes \psi) = d(\phi) \cup d(\psi)$
(A3) If $\phi \in \Phi$ and $V \in \mathscr{V}$ is such that $V \in d(\phi)$, then $d(\phi^{-V}) = d(\phi) \setminus \{V\}$
(A4) If $\phi \in \Phi$ and $V, W \in \mathscr{V}$ then $(\phi^{-V})^{-W} = (\phi^{-W})^{-V}$
(A5) If $\phi, \psi \in \Phi$ and $V \in \mathscr{V}$ is such that $V \notin d(\phi)$, then $(\phi \otimes \psi)^{-V} = \phi \otimes \psi^{-V}$.

Let $(\Phi, \mathscr{V}, d, \otimes, \mathrm{El})$ be a valuation algebra and $\{\phi_i\}_{i=0}^{m}$ a set of valuations in Φ such that $\bigcup_{i=0}^{m} d(\phi_i) = \mathscr{V}$. According to (A1), a *joint valuation* $\phi := \otimes_{i=0}^{m} \phi_i$ of this set can be defined with no ambiguities. According to (A2), $d(\phi) = \mathscr{V}$. According to (A4), the valuation obtained eliminating from ϕ all the variables of its domain is independent from the elimination sequence and can therefore be unequivocally denoted as $\phi^{-\mathscr{V}}$. According to (A3), $\phi^{-\mathscr{V}}$ is a valuation with empty domain and is called the *full marginal* of the joint valuation ϕ.

The complexity of the operation of variable elimination typically increases exponentially with the domain of the valuation considered. That often makes the computation of $\phi^{-\mathscr{V}}$ intractable, even if all the given valuations are defined on small domains.

However, (A5) suggests the possibility of eliminating some variable on a local domain, that is, without explicitly computing the joint valuation. This approach, called *fusion algorithm* [7], consists in the elimination of a variable only from the combination of the valuations such that the variable to eliminate is in their domain, i.e.:

$$\phi^{-V} = \left(\otimes_{i=0,\dots,m/V \notin d(\phi_i)} \phi_i \right) \otimes \left(\otimes_{j=0,\dots,m/V \in d(\phi_j)} \phi_j \right)^{-V} . \qquad (5)$$

The elementary procedure portrayed in (5) can be iterated over all the elements of $\mathscr{V}$, leading to $\phi^{-\mathscr{V}}$. According to (A4), any elimination sequence can be employed. Nevertheless, it should be pointed out how different sequences will require in general different computational times.

3 Hardness of CUR-based Classification

Call CCUR the problem to compute the undominated classes in a CUR-based classification problem with Bayesian nets. Let us initially focus on the binary version of the CCUR problem, that is, on a classification problem with only two classes, say c' and c''. We denote by CCURD the corresponding decision problem that involves deciding whether or not c' dominates c''. CCURD is clearly equivalent to (3), being 'true' (T) if (4) is greater than one and 'false' (F) otherwise. As a preliminary result, we will prove that CCURD is *coNP-complete*, i.e., the complement of an *NP-complete* problem [10]. In our proof, we take inspiration from the well-known result of Cooper [6], concerning probabilistic inference with Bayesian nets.

Recall that a decision problem $\mathscr{Q}$ is NP-complete if $\mathscr{Q}$ lies in the class NP and some known NP-complete problem $\mathscr{Q}'$ polynomially transforms to $\mathscr{Q}$ [11, p. 38]. In our case, we will transform a well known NP-complete problem, called 3-satisfiability (3SAT) [11], to the complement of CCURD. Let us recall the definition of 3SAT.

Let $\mathscr{U}$ be a collection of n Boolean variables. If U is a variable in $\mathscr{U}$ then u and $\neg u$ are said to be *literals* over $\mathscr{U}$. The literal u is true if and only if the variable U is true, while $\neg u$ is true if and only if the variable U is false. Let $\mathscr{K} = \{K_1, \dots, K_m\}$ be a non-empty collection of *clauses*, which are disjunctions of triples of literals, corresponding to different[3] variables of $\mathscr{U}$. The collection of clauses $\mathscr{K}$ over $\mathscr{U}$ is said to be *satisfiable* if and only if there exists a *truth assignment* for $\mathscr{U}$, that is, an assignment of Boolean values to the variables in $\mathscr{U}$, such that all the clauses in $\mathscr{K}$ are simultaneously true. The 3SAT decision problem involves determining whether or not there is a truth assignment for $\mathscr{U}$ such that $\mathscr{K}$ is satisfiable.

[3] This assumption is not included in the original transformation of the prototypical NP-complete problem SAT to 3SAT. Nevertheless, the transformation (see for example [11, p. 48]) does not require any clause to include literals corresponding to the same variable. Thus, also this version of 3SAT is NP-complete.

The NP-completeness of 3SAT can be used to prove the following:

Theorem 1 *CCURD is coNP-complete.*

Proof of Theorem 1 Given a generic 3SAT instance, say $\mathscr{U} = \{U_1, \ldots, U_n\}$ and $\mathscr{K} = \{K_1, \ldots, K_m\}$, we construct a Bayesian network such that $c' > c''$ if and only if $\mathscr{K}$ is not satisfiable. The nodes of the network correspond to the variables in $\mathscr{U}$, the clauses in $\mathscr{K}$ and the class C. The nodes corresponding to the clauses have four incoming arcs, three from the variables associated to the literals present in the definition of the clause and the fourth from the class node. The directed acyclic graph underlying the Bayesian network is therefore $\mathscr{G}(\mathscr{V}, \mathscr{E})$, with $\mathscr{V} = \{C, U_1, \ldots, U_n, K_1, \ldots, K_m\}$ and

$$\mathscr{E} = \{(U_{\alpha_{ij}}, K_j) \mid \begin{array}{c} i = 1, 2, 3, \\ j = 1, \ldots, m \end{array}\} \cup \{(C, K_j) \mid j = 1, \ldots, m\}, \tag{6}$$

where α_{ij} indexes the element of $\mathscr{U}$ corresponding to the i-th literal of the clause K_j. As an example, Figure 1 reports the graph corresponding to a 3SAT instance with three clauses and four variables in $\mathscr{U}$.

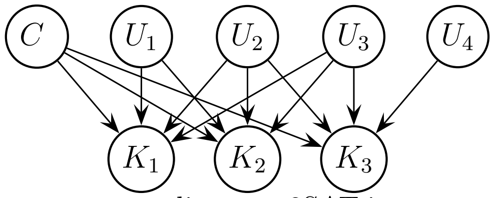

Fig. 1. A Bayesian net corresponding to a 3SAT instance with $\mathscr{U} = \{U_1, U_2, U_3, U_4\}$ and $\mathscr{K} = \{(u_1 \vee u_2 \vee u_3), (\neg u_1 \vee \neg u_2 \vee u_3), (u_2 \vee \neg u_3 \vee u_4)\}$.

Each node of $\mathscr{G}$ is assumed to represent a Boolean variable. The unconditional mass functions for the root nodes (i.e., the nodes without incoming arcs) are assumed to be uniform. Regarding the conditional mass functions we define them as in Table 1. Those values define a unique positive mass function for each clause and for every possible value of the parents of the clause.

c	$u_{\alpha_{1j}} \vee u_{\alpha_{2j}} \vee u_{\alpha_{3j}}$	$p(K_j = \mathrm{T} \mid c, u_{\alpha_{1j}}, u_{\alpha_{2j}}, u_{\alpha_{3j}})$
c'	T	2^{-2}
c''	T	2^{-1}
c'	F	2^{-1}
c''	F	$2^{-(m+1)}$

Table 1
Implicit definition of the conditional mass functions for the clause K_j, for each $j = 0, \ldots, m$. With an abuse of notation, $u_{\alpha_{ij}}$ denotes the i-th literal of K_j.

The directed acyclic graph $\mathscr{G}$, together with the specified mass functions, defines a Bayesian network. This is equivalent to a joint mass function, which assigns positive probability to every event. With respect to the evidence $E=e$ in the network, we suppose all the clauses in $\mathscr{K}$ are instantiated to the state 'true'. The remaining attribute variables, which are the variables in $\mathscr{U}$, are assumed to be missing. Expression (4) becomes:

$$\min_{\substack{u_j\in\{F,T\},\\ U_j\in\mathscr{U}}} \prod_{i=1}^{m} \phi_i(u_{\alpha_{1i}}, u_{\alpha_{2i}}, u_{\alpha_{3i}}), \tag{7}$$

where, for each $i = 1, \ldots, m$,

$$\phi_i(u_{\alpha_{1i}}, u_{\alpha_{2i}}, u_{\alpha_{3i}}) := \frac{p(K_i = \mathrm{T}|c', u_{\alpha_{1i}}, u_{\alpha_{2i}}, u_{\alpha_{3i}})}{p(K_i = \mathrm{T}|c'', u_{\alpha_{1i}}, u_{\alpha_{2i}}, u_{\alpha_{3i}})}. \tag{8}$$

Using the values of Table 1, the functions in (8) take the form:

$$\phi_i(u_{\alpha_{1i}}, u_{\alpha_{2i}}, u_{\alpha_{3i}}) = \begin{cases} 2^{-1} & \text{if } u_{\alpha_{1i}} \vee u_{\alpha_{2i}} \vee u_{\alpha_{3i}} = \mathrm{T} \\ 2^{m} & \text{otherwise.} \end{cases} \tag{9}$$

According to (9), if a clause is satisfied, the corresponding function attains its minimum value. Thus, if 3SAT is true, there exists a truth assignment over $\mathscr{U}$ satisfying all the clauses in $\mathscr{K}$, and all the functions (8) in (7) are simultaneously minimized. The minimum (7) is therefore 2^{-m} and the corresponding CCURD instance is false. If 3SAT is false, for all truth assignments at least one clause is violated and the corresponding function takes the value 2^m. That makes (7) always greater than one, because all the remaining $m-1$ functions cannot be less than 2^{-1}. Thus, CCURD is true.

This shows that each 3SAT instance is equivalent to an instance of the complement of CCURD; and we have achieved this by a transformation that is polynomial in the size of the 3SAT instance. Note, in addition, that the complement of CCURD is also in the class NP. A nondeterministic algorithm to solve the complement of CCURD has only to return a truth assignment for $\mathscr{U}$, provided that the corresponding value of the functions in (9) can be evaluated efficiently. It follows that the complement of CCURD is NP-complete and hence the thesis. □

As a direct consequence of Theorem 1, we can prove the following:

Corollary 2 *CCUR is NP-hard.*

Proof of Corollary 2 Let CCURD′ be the complement of CCURD. In order to prove the hardness of CCUR we consider a polynomial-time Turing reduction [11, p. 111] from CCURD′ to the binary version of CCUR. Suppose a hypothetical algorithm that solves instances of the binary CCUR problem is available. Let I be a CCURD$'$ instance that is true if c' does not dominate c'' and false otherwise. In order to solve such an instance we use the above algorithm for CCUR problems in the following way. If the algorithm yields c', then necessarily $c' > c''$, and I is false. If it yields both c' and c'', c' cannot dominate c'' and I is true. Analogously, if the algorithm yields only c'', I is still true. In any case, it turns out that a single call of the algorithm makes it possible to solve the CCURD$'$ instance I. Therefore CCURD$'$, which is NP-complete because of Theorem 1, is Turing reducible to the binary version of CCUR. This means that the binary version of CCUR is NP-hard, and, as a consequence, so is the general version. □

4 S-networks

The hardness result of the previous section establishes a limit to the possibility to compute classifications efficiently with CUR on Bayesian nets. Yet, efficient computation is possible on special classes of Bayesian networks: in fact, de Cooman and Zaffalon [4] provide a linear time algorithm to solve CCUR problems when the subgraph B^+, defined at the end of Section 2.2, is singly connected. In this paper we substantially extend such a result by providing a linear time algorithm that works in many cases also when B^+ is multiply connected.

The development of the new algorithm relies on the definition of a new kind of graphical model, called *s-network*, which allows us to abstract the main components of a CCUR problem.

4.1 Basic Definitions

Definition 3 *Let $\mathscr{G}$ be a directed acyclic graph in which some nodes, say $A_0, \ldots, A_m$ ($m \geq 0$), are marked as* special nodes *(or* s-nodes*) such that every arc of $\mathscr{G}$ has a special node as second endpoint. Each node of $\mathscr{G}$ is identified with a variable that takes finitely many values. Every special node A_i in $\mathscr{G}$ ($i = 0, \ldots, m$) is associated with a so-called potential $\phi_i(A_i^+)$, defined for all the values of its argument. A_i^+ is the vector variable (A_i, Π_{A_i}), with generic value a_i^+, where Π_{A_i} are the parents of A_i. The graph $\mathscr{G}$, together with the collection of potentials $\{\phi_i\}_{i=0}^m$, is called* s-network.

Given an s-network $\mathscr{G}$, its *minimum* is defined by

$$\min_{\substack{a_j^+ \in \mathscr{A}_j^+,\\ j\in\{0,\dots,m\}}} \prod_{i=0}^{m} \phi_i(a_i^+). \tag{10}$$

Note that Definition 3 does not exclude the case of disconnected s-networks. If $\mathscr{G}_k$ is a connected component of a (disconnected) s-network $\mathscr{G}$, we can regard $\mathscr{G}_k$, together with the potentials of $\mathscr{G}$ corresponding to the s-nodes of $\mathscr{G}_k$, as an s-(sub)network. The following result holds:

Theorem 4 *Let $\mathscr{G}$ be a disconnected s-network. The minimum of $\mathscr{G}$ factorizes in the product of the minima of the s-networks corresponding to the connected components of $\mathscr{G}$ with at least one s-node.*

In the next section, we show that by calculating the minima of s-networks we can solve CCUR instances.

4.2 Minima of S-networks solve CCURD Problems

Let I be a CCURD instance that involves deciding whether or not $c' > c''$. We denote by $\mathscr{G}_I$ the directed graph obtained from B^+ marking as special $C = A_0$ together with its children, removing the arcs that leave C and the observed nodes, and removing the observed nodes that are not special. The following algorithm is an obvious (linear time) implementation of this transformation:

Algorithm 1 *An algorithm to build up a graph $\mathscr{G}_I(\mathscr{V},\mathscr{E})$ given a CCURD (or CCUR) instance I.* $\mathrm{T}(\epsilon)$ *represents the first endpoint of the arc ϵ, while E is the subset of the observed attribute variables of I.*

```
1   $\mathscr{G}_I := B^+$;
2   **foreach** $V \in \mathscr{V}$ {
3         **if** $V = C$ **or** $C$ parent of $V$ {
4               mark $V$ as special; }}
5   **foreach** $\epsilon \in \mathscr{E}$ {
6         **if** $\mathrm{T}(\epsilon) \in E$ **or** $T(\epsilon) = C$ {
7               remove $\epsilon$; }}
8   **foreach** $V \in E$ {
9         **if** $V$ not special {
10              remove $V$; }}
```

Each node of $\mathscr{G}_I$ is identified with a variable that takes finitely many values, as follows. The target node A_0 and the nodes of $\mathscr{G}_I$ corresponding to the observed attribute variables of I are assumed to be constants, i.e., their possibility spaces contain a single value, while the remaining nodes, which are the missing attribute variables in I, are identified with the same categorical variables of the original problem. Finally, we set:

$$\phi_0(a_0^+) := \frac{p(c'|\pi_C)}{p(c''|\pi_C)} \tag{11}$$

$$\phi_i(a_i^+) := \frac{p(a_i|\pi'_{A_i})}{p(a_i|\pi''_{A_i})} \qquad i = 1, \ldots, m. \tag{12}$$

The graph $\mathscr{G}_I$ together with the potentials as in (11) and (12) can be easily recognized to be an s-network. The computation of the minimum of this s-network solves the original CCURD instance, according to the following:

Theorem 5 *I is true if and only if the minimum of the s-network $\mathscr{G}_I$ is greater than one.*

As a numerical example, let us consider a Bayesian network over the boolean variables $(A_0, \ldots, A_6)$ with the graphical structure displayed in Figure 2. Let $C := A_0$ be the class variable and c' and c'' the possible classes.

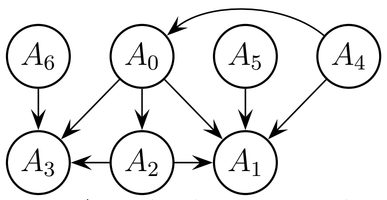

Fig. 2. A multiply connected Bayesian network.

We assume uniform unconditional mass functions for the root nodes, while Tables 2, 3, 4 and 5 specify the conditional mass functions for the remaining nodes.

a_4	$p(C = c'\|a_4)$
T	0.8
F	0.9

Table 2
Conditional mass functions for node C.

The decision whether c' dominates c'' or not, assuming all the attribute variables $(A_1, \ldots, A_6)$ to be missing, can be regarded as a CCURD instance I. First, we use Algorithm 1 to construct the graph $\mathscr{G}_I$ corresponding to the instance I. The result is the s-network displayed in Figure 3.

According to the procedure described in this section, each node of $\mathscr{G}_I$ is identified with the same boolean variable of the original Bayesian network, except A_0 that is assumed to be constant. Furthermore, we can use the probability

c	a_2	a_4	a_5	$p(A_1 = \text{T}\|c, a_2, a_4, a_5)$
c'	T	T	T	0.4
c'	T	T	F	0.2
c'	T	F	T	0.3
c'	T	F	F	0.1
c'	F	T	T	0.7
c'	F	T	F	0.9
c'	F	F	T	0.8
c'	F	F	F	0.1
c''	T	T	T	0.2
c''	T	T	F	0.3
c''	T	F	T	0.3
c''	T	F	F	0.2
c''	F	T	T	0.4
c''	F	T	F	0.9
c''	F	F	T	0.7
c''	F	F	F	0.2

Table 3
Conditional mass functions for node A_1.

c	$p(A_2 = \text{T}\|c)$
c'	0.4
c''	0.7

Table 4
Conditional mass functions for node A_2.

c	a_2	a_6	$p(A_3 = \text{T}\|c, a_2, a_6)$
c'	T	T	0.6
c'	T	F	0.7
c'	F	T	0.2
c'	F	F	0.8
c''	T	T	0.2
c''	T	F	0.9
c''	F	T	0.2
c''	F	F	0.4

Table 5
Conditional mass functions for node A_3.

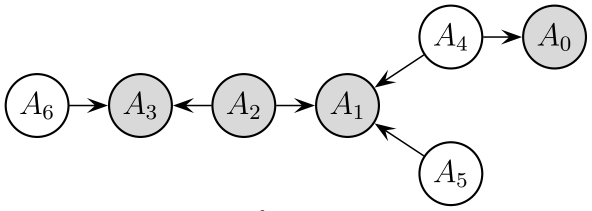

Fig. 3. The s-network $\mathscr{G}_I$ returned by the application of Algorithm 1 to a CCURD instance I on the Bayesian network of Fig. 2. The s-nodes are displayed in gray.

specifications in Tables 2, 3, 4 and 5 to define a potential for each special node of $\mathscr{G}_I$ as in (11) and (12). Finally, according to Theorem 5, the computation of the minimum of $\mathscr{G}_I$ solves the CCURD instance I.

Theorem 5 is the basis to solve also a class of CCUR problems. Let us therefore consider a generic classification problem with missing data, whose set of classes is $\mathscr{C} := \{c_1, \ldots, c_r\}$. For each pair of classes, we can consider the corresponding binary CCUR instance. For each binary CCUR instance, we consider two CCURD instances as follows. If the binary CCUR instance requires to compare the classes between c_i and c_j, the first CCURD instance checks whether or not $c_i \triangleright c_j$, while the second checks $c_j \triangleright c_i$. Whenever one of these CCURD instances is true, the dominated class is rejected. The following algorithm reports the full procedure detecting the optimal classes:

Algorithm 2 *The procedure to solve a CCUR instance with set of classes $\mathscr{C} := (c_1, \ldots, c_r)$. The output is the set of the optimal classes $\mathscr{C}_{opt}$.*

```
1  $\mathscr{C}_{opt} := \mathscr{C}$;
2  for $i = 1, \ldots, r$ {
3      for $j = 1, \ldots, r$ {
4          if $i < j$ {
5              if $c_i \triangleright c_j$ { remove $c_j$ from $\mathscr{C}_{opt}$;}
6              if $c_j \triangleright c_i$ { remove $c_i$ from $\mathscr{C}_{opt}$;}}}}
7  return $\mathscr{C}_{opt}$;
```

Concerning the computational complexity of Algorithm 2, the total number of solved CCURD instances is quadratic in the input size, being exactly $r \cdot (r-1)$.

5 Solving Problems on S-networks

In this section we show that the minimum of an s-network can be regarded as a full marginal of a joint valuation in a VA, as defined in Section 2.3. Furthermore, the fusion algorithm (5) can be employed to minimize s-networks. In the special case of s-polytrees, the algorithm takes linear time for an appropriate elimination sequence.

5.1 Minima of S-networks as Local Computations on Valuation Algebras

We firstly introduce the following:

Theorem 6 *Let $\mathscr{G}$ be an s-network. Let $\mathscr{V}$ be the nodes of $\mathscr{G}$ and Φ the set of all the nonnegative real functions of any possible subset of $\mathscr{V}$. Let d be the map returning the variables in the argument of those functions and $\otimes$ the pointwise function product. Let also* El *be a variable elimination defined as $\phi^{-A_i} := \min_{a_i \in \mathscr{A}_i} \phi$. Thus, $(\Phi, \mathscr{V}, d, \otimes, \mathrm{El})$ is a valuation algebra and the potentials of $\mathscr{G}$, say $\{\phi_i\}_{i=0}^m$, are valuations in Φ.*

Accordingly, we can express the minimum of $\mathscr{G}$ as follows:

Theorem 7 *Let $\mathscr{G}$ be an s-network, $\{\phi_i\}_{i=0}^m$ its potentials and $(\Phi, \mathscr{V}, d, \otimes, \mathrm{El})$ the corresponding VA as in Theorem 6. Then, $\min \mathscr{G} = (\phi_0 \otimes \cdots \otimes \phi_m)^{-\mathscr{V}}$.*

The fusion algorithm can therefore be used to calculate the minimum of an s-network. In general, the computation will take exponential time. Nevertheless, for a particular topology of the s-network $\mathscr{G}$, and an appropriate choice of the ordering in which the variables of $\mathscr{V}$ are eliminated, the algorithm becomes efficient. That is shown in the following.

5.2 Nodes Sorting on S-polytrees

We call *s-polytree* an s-network $\mathscr{G}$ such that the underlying graph is a polytree. As an example, the s-network in Figure 3 is an s-polytree. The set $\mathscr{V}$ of the nodes of an s-polytree $\mathscr{G}$ has a natural structure of metric space. Given two nodes U and V, there is a single undirected path connecting them. Let $\delta(U, V)$ be the number of edges making up this path. The map δ is clearly a metric over $\mathscr{V}$ and $\delta(U, V)$ is said to be the *distance* between U and V. Let us call *neighbors* of U the nodes of $\mathscr{V}$ at distance one from U.

Given an s-polytree $\mathscr{G}$, an s-node A_k of $\mathscr{G}$ is said to be *lonely* if there is a node U of $\mathscr{G}$ such that A_k is the s-node at maximum distance from U (or one of them, if there are many). As an example, in the s-polytree of Fig. 3, A_0 is the s-node at maximum distance from A_6 and can therefore be regarded as a lonely node. The lonely nodes of an s-polytree can be characterized as follows:

Theorem 8 *Let $\mathscr{G}$ be an s-polytree with at least two s-nodes and A_k a lonely node of $\mathscr{G}$. The variables in A_k^+, with the exception of a single variable S, appear only in the argument of ϕ_k.*

As an example, in the case of the s-polytree of Fig. 3, $A_0^+ = (A_0, A_4)$, and while A_0 appears only in the argument of ϕ_0, A_4 appears also in ϕ_1.

Given a lonely node A_k, we denote by $\tilde{A}_k^+$ the vector variable that includes all the variables in A_k^+ except S and we refer to these variables as the *extreme leaves* of the s-polytree $\mathscr{G}$ with respect to A_k. In the case of s-polytrees with a single s-node, all the nodes are extreme leaves.

In the example of Figure 3, A_0 is the only extreme leave of $\mathscr{G}$ with respect to A_0 itself.

An s-node A_l is said to be a *conjugate* node of a lonely node A_k, if the variable $S \in A_k^+$, which is not included in $\tilde{A}_k^+$, appears also in A_l^+. We can therefore regard S as the intersection of A_k^+ and A_l^+.

For example, A_1 is clearly the only conjugate of A_0 in the s-polytree of Figure 3, and A_4 can be regarded as the intersection between $A_0^+ = (A_0, A_4)$ and $A_1^+ = (A_1, A_2, A_4, A_5)$.

Call *siblings* two distinct children of the same parent. The conjugate nodes of a lonely node are characterized by the following:

Theorem 9 *Let A_k be a lonely node of an s-polytree $\mathscr{G}$ with at least two s-nodes. The conjugate nodes of A_k are the special neighbors and the siblings of A_k. Furthermore, A_k has at most a special neighbor; and if no s-nodes lie in the neighborhood of A_k, then A_k has at least one sibling.*

In the case of the s-polytree of Figure 3, A_0 has no special neighbor and its unique special sibling A_1 is the only conjugate of A_0.

As a consequence of Theorem 9, for each lonely node A_k, there is at least a conjugate A_l.

It is indeed possible to show that the *pruning* of the extreme leaves of $\mathscr{G}$ preserves the s-polytree structure, as stated in the following:

Theorem 10 *Let $\mathscr{G}$ be an s-polytree with at least two special nodes, and A_k a lonely node of $\mathscr{G}$. If A_k is marked as not special, the nodes in $\tilde{A}_k^+$ are removed from $\mathscr{G}$, and ϕ_k is dropped from $\{\phi_i\}_{i=0}^m$, then a new s-polytree $\mathscr{G}'$ is obtained.*

As an example, the s-polytree of Figure 3 becomes a new s-polytree with three s-nodes after the pruning of A_0 (and the removal of ϕ_0). A lonely node in a pruned s-polytree $\mathscr{G}'$ can be characterized by the following:

Theorem 11 *Let $\mathscr{G}$ be an s-polytree. Given an arbitrary node of $\mathscr{G}$, say U, let A_k and $A_{k'}$ be respectively the first and the second s-nodes at maximum distance from U (or one of them, if there are many). Let $\mathscr{G}'$ be the s-polytree obtained marking A_k as not special, removing the nodes in $\tilde{A}_k^+$ from $\mathscr{G}$, and ϕ_k from the set of potentials, as in Theorem 10. Thus, $A_{k'}$ is a lonely node of $\mathscr{G}'$.*

In the case of the s-polytree $\mathscr{G}$ of Figure 3, A_1 is the second s-node, after A_0, at maximum distance from A_6 and can therefore be regarded as a lonely node of the pruned s-polytree $\mathscr{G}'$, obtained removing A_0 and ϕ_0 according to Theorem 10.

This pruning procedure described in Theorem 10 yields an s-polytree and can therefore be iterated until an s-polytree with a single s-node is returned. As a consequence of Theorem 11, if we sort the s-nodes of $\mathscr{G}$ according to their distance from U, the i-th element of this sequence is a lonely node of the s-polytree returned by the i-th iteration of the pruning procedure. For each node of this sequence, we can consider the corresponding extreme leaves. It is trivial to check that all the elements of $\mathscr{V}$ appear in this collection of extreme leaves.

The discussion in this section suggests the opportunity to employ this collection of extreme leaves as an elimination sequence for the variables in $\mathscr{V}$ in order to minimize s-polytrees through the fusion algorithm. It is finally clear that, in the case of s-polytrees with a single s-node, there is a single potential and therefore no particular strategy to detect an efficient elimination sequence is required.

5.3 Solution Algorithm

If $\mathscr{G}$ is an s-polytree and A_k is a lonely node of $\mathscr{G}$, the elimination of the extreme leaves of $\mathscr{G}$ with respect to A_k can be restricted to ϕ_k because of Axiom (A5). Thus:

$$(\phi_0 \otimes \ldots \otimes \phi_m)^{-\tilde{A}_k^+} = (\otimes_{i=0,\ldots,m/i\neq k}\phi_i) \otimes \phi_k^{-\tilde{A}_k^+}. \tag{13}$$

But $d(\phi_k^{-\tilde{A}_k^+})$, that is a single variable because of Theorem 8 and Axiom (A2), appears also in ϕ_l, where A_l is a conjugate of A_k, by definition of conjugate. With a simple redefinition of the potential of A_l:

$$\phi_l = \phi_l \otimes \phi_k^{-\tilde{A}_k^+}, \tag{14}$$

the information associated to the potential ϕ_k after the elimination of the variables in $\tilde{A}_k^+$ can be embedded in ϕ_l. Notably, $d(\phi_l') = d(\phi_l)$ because of Axiom (A2), that means that the potential redefinition in Equation (14) does not affect the domain of ϕ_l. Finally, if we drop the potential ϕ_k from the set of potentials, a new s-polytree $\mathscr{G}'$ is obtained because of Theorem 10 and the procedure can be iterated. The overall procedure, returning the minimum of the s-polytree because of Theorem 7, is reported in the following algorithm:

Algorithm 3 *The findMin routine. In input we have an s-polytree $\mathscr{G}$. The subroutine findInter returns a vector with the intersection of two arrays of variables.*

```
1    U := randomly chosen node of $\mathscr{G}$;
2    $(\delta_0,\ldots,\delta_m)$ := findDistances($\mathscr{G}$,U);
3    while number of s-nodes in $\mathscr{G} > 1$ {
4        $k := \arg\max_j \delta_j$;
5        $A_l$ := findConjugate($A_k$,$\mathscr{G}$);
6        S := findInter($A_k^+$,$A_l^+$);
7        $\tilde{A}_k^+$ := remove S from $A_k^+$ ;
8        $\phi_l = \phi_l \otimes \phi_k^{-\tilde{A}_k^+}$;
9        mark $A_k$ as not special;
10        drop the nodes in $\tilde{A}_k^+$ from $\mathscr{G}$;
11        remove $\phi_k$, from $(\phi_0, \ldots, \phi_m)$;
12        remove $\delta_k$, from $(\delta_0, \ldots, \delta_m)$;}
13   return $\phi_l^{-d(\phi_l)}$;
```

The distances between a randomly chosen node U and the s-nodes of $\mathscr{G}$ are initially computed (lines 1–2 of Alg. 3). The subroutine *findDistances*($\mathscr{G}$,U), returning the distances between U and the s-nodes of $\mathscr{G}$, can be implemented through the well known *depth first search* (DFS) algorithm [12] over the undirected graph obtained forgetting the orientation of the arcs of $\mathscr{G}$.

A lonely node A_k of $\mathscr{G}$ can therefore be detected as the s-node at maximum distance from U (line 4). Algorithm 4, detects a conjugate A_l of A_k (line 5) and the extreme leaves of $\mathscr{G}$ with respect to A_k are therefore obtained (line 6–7). These variables are indeed eliminated and the result is embedded on ϕ_l as in (14) (line 8). Finally (lines 9–12), $\mathscr{G}$ is transformed by the pruning procedure of Theorem 10 in a new s-polytree with an s-node fewer. The overall procedure is iterated (line 3) until an s-polytree with a single s-node, whose minimization is trivial (line 13), is returned.

Algorithm 4 *The findConjugate function. The inputs are the polytree $\mathscr{G}$ and a lonely node A_k. The output findConjugate($\mathscr{G}$,A_k) is a conjugate of A_k. The subroutine findNeighbors returns the neighbors of the node in its argument.*

```
1   foreach $V \in$ findNeighbors($A_k$) {
2       if $V$ is special {
3           $A_l := V$;
4           go to 8; }
5       else {
6           if $V$ has a children $W$ {
7               $A_l := W$; }}}
8   return $A_l$;
```

As an example, Algorithm 3 can be used to calculate the minimum of the s-polytree of Fig. 3. The distances between $U := A_6$ and the s-nodes of $\mathscr{G}$ are: $\delta_0 = 5$, $\delta_1 = 3$, $\delta_2 = 2$, $\delta_3 = 1$. Thus, A_0 is a lonely node of $\mathscr{G}$ and its sibling A_1 is a conjugate of it.

Clearly, $\tilde{A}_0^+ = A_0$ and the redefinition of ϕ_1 should be:

$$\phi_1(a_1, a_2, a_4, a_5) = \phi_1(a_1, a_2, a_4, a_5) \cdot \min_{a_0 \in \mathscr{A}_0} \phi_0(a_0, a_4). \tag{15}$$

Furthermore, A_0 is marked as not-special and dropped from $\mathscr{G}$, the potential ϕ_0 is removed from $\{\phi_i\}_{i=0}^3$, and similarly δ_0 is removed from $\{\delta_i\}_{i=0}^3$. After these operations, $\mathscr{G}$ is now an s-polytree with three s-nodes. A_1 is clearly its s-node at maximum distance from A_6 and it is therefore a lonely node of $\mathscr{G}$, while A_2 is a conjugate of it. The extreme leaves are A_1, A_4 and A_5 and the redefinition of ϕ_2 is:

$$\phi_2(a_2) = \phi_2(a_2) \cdot \min_{a_1 \in \mathscr{A}_1, a_4 \in \mathscr{A}_4, a_5 \in \mathscr{A}_5} \phi_1(a_1, a_2, a_4, a_5). \tag{16}$$

A further iteration of the procedure yields to:

$$\phi_3(a_2, a_3, a_6) = \phi_3(a_2, a_3, a_6) \cdot \phi_2(a_2), \tag{17}$$

and finally, we conclude that the minimum of the s-polytree $\mathscr{G}_I$ is:

$$\min_{a_3 \in \mathscr{A}_3, a_2 \in \mathscr{A}_2, a_6 \in \mathscr{A}_6} \phi_3(a_2, a_3, a_6) = \frac{2}{3}. \tag{18}$$

According to Theorem 5, the CCURD instance I associated to $\mathscr{G}_I$ is therefore false and c' does not dominate c''.

Now, let $\overline{I}$ be the CCURD instance involving the decision whether or not $c'' > c'$ with all the attribute variables missing. We can proceed in complete analogy with the procedure used to solve I. The numerical value of the minimum of $\mathscr{G}_{\overline{I}}$ is $\frac{4}{189}$. $\overline{I}$ is therefore false and we conclude that the two classes are mutually undominated. Therefore, if all the attribute variables are missing, we are not able to identify a single optimal class and both the values c' and c'' are plausible.

Finally, to detect whether or not Algorithm 3 can be used to solve a given CCURD instance I, it is sufficient to check if the graph $\mathscr{G}_I$ returned by Algorithm 1 is a polytree. The condition $|\mathscr{V}| = |\mathscr{E}| + 1$ for $\mathscr{G}_I(\mathscr{V}, \mathscr{E})$ can therefore be used as an obvious applicability check. Note that Algorithm 1 obtains $\mathscr{G}_I$ removing some nodes and arcs from B^+. Therefore $\mathscr{G}_I$ can be a polytree also if the original Markov blanket is multiply connected (e.g., the net in Figure 2).

Remember that we are focusing on connected s-networks. In the general case of a disconnected s-network $\mathscr{G}$, we have only to check whether or not the graph is a polyforest. In the positive case, Algorithm 3 can be used to calculate the minima of the s-polytrees associated to the connected component of $\mathscr{G}$ with at least one s-node, while the overall minimum is just the product of these minima because of Theorem 4.

Finally, concerning the efficiency of the overall procedure:

Theorem 12 *Algorithm 3 has linear complexity.*

Note that in analogy with [4, Section 6], the common technique called *loop cutset conditioning* can be used to solve a CCUR instance I even if the graph

$\mathscr{G}_I$ returned by Algorithm 1 is not a polyforest. In this case the computation will take exponential time.

6 Some comments about CUR-based classification

So far we have focused on algorithms for CUR-based classification with Bayesian networks. In this section, we would like to give a broader perspective of this approach so as to clarify its characteristics and possible usages.

An important point concerns the cautiousness of CUR. Remember that CUR assumes near-ignorance about the missingness process, and this implies having to consider all the completions of the missing values as part of the updating rule. Not surprisingly, this procedure is likely to yield partially indeterminate conclusions (i.e., classifications), especially when there are missing attribute variables that are important to predict the class. Avoiding indeterminacy is therefore tightly connected with being able to measure all good predictors. This will probably not be the case at the initial stages of interaction with an expert system, in which only some of the variables are measured. But the interaction is often a dynamic rather than a static process (this is very natural with credal classifiers, and more generally with imprecise probability models) in which more and more measures are collected along the way towards definite conclusions. This dynamic way of using expert systems would eventually lead CUR to yield strong enough conclusions, with the advantage of having the intermediate conclusions, guiding the process, not biased by potentially strong assumptions about the missingness process.

Obtaining stronger conclusions would be favored also by modifying CUR in such a way that it may apply to incomplete rather than only to missing observations. An incomplete observation is defined as a set-based observation that does not necessarily coincide with the entire possibility space (as with missing data); the fact that some values may be excluded obviously favors obtaining stronger conclusions (for an example of expert system that exploits incomplete observations see [13]). It is worth pointing out that the algorithms presented in this paper for CUR can be immediately extended to incomplete observations of attributes: whenever there is a minimization, it is sufficient for the extension to minimize over the observed subset an attribute's possibility space rather than over the entire space. It should also be noted that the evolution of CUR towards incomplete observations has already been proposed under the name of *conservative inference rule* [14], which actually extends CUR under more substantial respects, for instance by establishing the theoretical underpinning for statical applications of these conservative rules.

Given that the last observation points to possible uses of CUR in a statistical pattern classification context, it may be useful to briefly discuss the topic, also if statistical applications are out of the scope of the present paper. One important thing to be aware of is that rules such as CUR find justification in a statistical classification setting that produces (complete) data in an independently and identically distributed way, when the missingness process is *not* identically distributed, i.e., when each unit of (complete) data may be subject to a different missingness process. [4] Interestingly, in such a setup traditional precise (i.e., non-credal) classifiers cannot be really considered competitors of credal classifiers when the missingness processes is (partly) unknown: it is very easy to build applications that make every precise classifier fail to predict the right classes; and this may be even done so that there is no way to know such bad performance in advance by making the empirical tests traditionally employed in the classification practice (see [14, Sect. 6]). The considered setting seems therefore particularly suited for CUR-based classification and its extensions, and worth exploring.

7 Conclusions

Probabilistic expert systems suggest actions on the basis of the available evidence about a domain. Often such an evidence is only partial, due to a number of reasons such as economic or time constraints. In order for the suggested actions to be credible, it is important to properly take into account the process that makes the evidence partial by hiding the state of some of the variables used to describe the domain. The recently derived conservative updating rule achieves this by considering a near-ignorance about the missingness process, and by updating beliefs accordingly. In order to make the rule profitably used it is important to develop efficient algorithms to compute with it.

In this paper we have shown that it is not possible in general to create efficient algorithms for such a purpose (unless P=NP): in fact, using the conservative updating rule to do efficient classification with Bayesian networks is shown to be NP-hard. This parallels analogous results with more traditional ways to do classification with Bayesian nets: in those cases, the computation is efficient only on polyforest-shaped Bayesian networks. Our second contribution shows that something similar happens using the conservative updating, too. Indeed we provide a new algorithm for robust classification that is efficient on polyforest-shaped s-networks. This extends substantially a previously existing algorithm which, loosely speaking, is efficient only on disconnected s-networks.

Yet, it is important to stress that the computational difference between tra-

[4] In fact, if the missingness process is also (independently and) identically distributed, a more traditional approach should be employed [14, Sect. 5].

ditional classification with Bayesian nets and robust classification based on the conservative updating rule is remarkable: first, the former is based on the entire net, while the latter only on the net made by the class variable with its Markov blanket; second, while the former needs that the entire network is a polyforest in order to obtain efficient computation, the latter requires only that the associated s-network is. This means that the computation will be efficient also in many cases when the class variable with its Markov blanket forms a multiply connected net in the original Bayes network. In other words, computing robust classifications with the conservative updating will be typically much faster than computing classifications with the traditional updating rule. Given that the latter classifications are necessarily included in the former, by definition of the conservative updating rule, it seems to be worth considering robust classifications not only as a stand-alone task, but also as a pre-processing step of traditional classification with Bayesian nets.

With respect to future research, a natural development would be a generalization of our algorithms to the conservative inference rule [14], which models also an hybrid situation of near-ignorance about missingness process of some variables and MAR condition satisfied by the others. It seems also possible to proceed as in [4, Sect. 7] to employ our algorithm also in the case of *credal networks* [15], which are graphical models extending the formalism of Bayesian networks by allowing sets of mass functions. Some recent works [16] seem to be promising highly for a development in these directions.

A Proofs and lemmas

Lemma 13 *Let A_k and A_l be two distinct s-nodes of an s-network $\mathscr{G}$. A_k^+ and A_l^+ can share some variables if and only if A_k is a parent or a child or a sibling of A_l.*

Proof of Lemma 13 Let S be a variable included both in $A_k^+ = (A_k, \Pi_{A_k})$ and $A_l^+ = (A_l, \Pi_{A_l})$. We distinguish the four possible cases: (i) $S = A_k = A_l$. (ii) $S = A_k$ and $S \in \Pi_{A_l}$ (iii) $S = A_l$ and $S \in \Pi_{A_k}$ (iv) $S \in \Pi_{A_k}$ and $S \in \Pi_{A_l}$.

The first case cannot take place because A_k and A_l are assumed to be distinct nodes. In the second case A_k is clearly a parent of A_l, while, *vice versa*, A_l is parent of A_k in the third case. Finally, A_k and A_l are sibling through their common parent S in the fourth case.

On the other hand, if A_k is a parent (child) of A_l, clearly A_k^+ shares the variable A_k (A_l) with A_l^+. Finally, if A_k and A_l are siblings, their common parents appear both in A_k^+ and A_l^+. $\square$

Proof of Theorem 4 Let $(\mathscr{G}_1, \ldots, \mathscr{G}_s)$ be the connected components of $\mathscr{G}$ with at least one s-node. We denote as M_i the vector of the indices of the s-nodes that are in $\mathscr{G}_i$ $(i = 1, \ldots, s)$. Clearly, $(M_1, \ldots, M_s)$ represents a partition of $M := \{0, \ldots, m\}$.

For each $k \in M_i$ and $l \in M_j$ $(i, j = 1, \ldots, s$ and $i \neq j)$, A_k^+ and A_l^+ cannot share any variable because of Lemma 13. The minimum of $\mathscr{G}$ can therefore be expressed as a product of local minima $\prod_{k=1}^{s} \mu_k$, where, for each $k = 1, \ldots, s$:

$$\mu_k := \min_{\substack{a_j^+ \in \mathscr{A}_j^+,\\ j \in M_k}} \prod_{i \in M_k} \phi_i(a_i^+). \tag{A.1}$$

But (A.1) is the minimum of the s-(sub)network $\mathscr{G}_k$, that proves the thesis. $\square$

Proof of Theorem 5 Using (11) and (12), the minimum of $\mathscr{G}_I$ becomes:

$$\min_{\substack{a_j \in \mathscr{A}_j,\\ j=\{0,\ldots,n\}}} \left[\frac{p(c'|\pi_C)}{p(c''|\pi_C)} \cdot \prod_{i=1}^{m} \frac{p(a_i|\pi'_{A_i})}{p(a_i|\pi''_{A_i})} \right]. \tag{A.2}$$

The missing attribute variables of the CCURD instance I are exactly the non-constant variables in (A.2), while the constant variables have the same values of the observed attribute variables in I. Finally, as observed in [4, Sect. 6], (3) is preserved by dropping the arcs leaving the nodes in the subset of the observed nodes E for each $c \in \mathscr{C}$ and $r \in \mathscr{R}$. Thus, (A.2) coincides with the expression (4) relative to I. That proves the thesis. $\square$

Proof of Theorem 6 It is obvious to see that the operations of labelling, combination and variable elimination defined as in the statement of Theorem 6 are well defined according to the definition of VA in Section 2.3. In order to prove the theorem, it is therefore sufficient to check that the five axioms are satisfied. The commutativity and associativity of $\otimes$ naturally comes from the same property satisfied by the pointwise product between function and (A1) is therefore satisfied. It is also obvious to observe that the argument of a product of two functions is the union of the arguments of the functions and therefore also (A2) holds. The argument of $\phi^{-A_i} = \min_{a_i \in \mathscr{A}_i} \phi$ is clearly the argument of ϕ deprived by A_i. Thus, also (A3) is satisfied. Regarding (A4), the minimization of a function over two variables on its arguments are independent from the order of minimization and also this axiom holds. if ψ and ϕ are two valuations and A_i is only in the argument of ψ, then the minimization over A_i of the product of this two functions is the product between ϕ and the minimum of ψ. That means $(\psi \otimes \phi)^{-A_i} = \psi \otimes \phi^{-A_i}$, i.e., also (A5) holds. $\square$

Proof of Theorem 7 It is sufficient to rewrite the products between potentials in (10) as combinations, according to the definition in the statement of Theorem 6, and the minimization as a (full) variable elimination. $\square$

Lemma 14 *Let $\mathscr{G}$ be an s-polytree with at least two s-nodes. Let A_k be a lonely node of $\mathscr{G}$. Then the following holds:*

(i) *A_k has at most a special neighbor.*
(ii) *A_k can have special siblings, but all these siblings have a single parent in common with A_k, that is the same for all of them.*
(iii) *If A_k has actually a special neighbor A_l, the possible special siblings of A_k should have in A_l the single parent in common with A_k.*

Proof of Lemma 14 Let U be a node of $\mathscr{G}$ such that A_k is the s-node of $\mathscr{G}$ (or one of them, if there are many) at maximum distance from U. The undirected path from U to A_k is unequivocally determined, because $\mathscr{G}$ is a polytree. Clearly $U \neq A_k$, because otherwise another s-node of $\mathscr{G}$ would be more distant from U than A_k. Therefore, the path includes at least two nodes. Let S be the node preceding A_k in the path.

With the only possible exception of S, the neighbors of A_k cannot be special. If A would be a special neighbor of A_k different from S, the undirected path between U and A would cross A_k and A would be an s-node more distant from U than A_k. That proves (i).

If A_k has special siblings, all these s-nodes should have S as common parent. If A would be a special sibling of A_k through a common parent different from S, A would be an s-node more distant from U than A_k. That proves that S is the only parent common to A_k and its special siblings, as stated by (ii).

Finally, it was already proved that, if A_k has a special neighbor, this is S. Therefore the parent common to the special siblings of A_k, if actually A_k has a special neighbor, is exactly this neighbor. That proves (iii). $\square$

Proof of Theorem 8 Let us first consider the case where A_k has not special neighbors. According to Lemma 13, A_k^+ can share some variables only with the vector variables corresponding to the possible special siblings of A_k. As a consequence of (ii) in Lemma 14, all the siblings of A_k have a single parent in common with A_k, that is the same for all of them. Let S be this node. S is clearly the only variable of A_k^+ that can appear also in some other vector variable.

Otherwise, if A_k has some special neighbor, then this is unique because of (i) in Lemma 14. Let A_l be this s-node. Point (iii) in Lemma 14 states that, if A_k has some special sibling, A_l should be a parent of A_k and also parent of all these siblings. Therefore, if A_l is child of A_k, A_k cannot have special siblings. In this case, A_k^+ can share some variable only with A_l^+ and, clearly, the only shared variable is A_k.

Finally, if A_l is parent of A_k, A_k can have some special sibling. We have already observed that A_l should be parent of all these siblings. Lemma 13 states that A_k^+ can share its variables only with A_l^+ and with the vector variables associated to the possible special siblings of A_k. In any case, A_l is the only variable of A_k^+ appearing also in some other vector variable. □

Proof of Theorem 9 All the special neighbors and the special siblings of A_k are conjugate nodes of A_k because of Lemma 13. On the other side, if A_k is a lonely node of $\mathscr{G}$ and A_l a conjugate node of A_k, then A_k^+ and A_l^+ should share some variable. Thus, always because of Lemma 13, A_k and A_l are neighbors or siblings.

Furthermore, A_k has at most a special neighbor because of (i) in Lemma 14.

If the neighbors of A_k are all non-special, they all should be parents of A_k. The reason is that the arcs of an s-network cannot terminate on a non-special node. For the same reason those non-special parents of A_k cannot have any parent. Nevertheless, at least one of them should have a child, because otherwise $\mathscr{G}$ would include only a single s-node. This child is a second endpoint of an arc of an s-network and it is therefore a special node. Let A_l be this s-node. Clearly, A_l is a special sibling of A_k. That proves that a lonely node with no special neighbors should have at least one special sibling. □

Lemma 15 *Let $\mathscr{G}$ be an s-polytree with at least two s-nodes. Let A_k be a lonely node of $\mathscr{G}$. If A_k has a special neighbor, the non-special parents of A_k are leaf nodes of the undirected tree obtained from $\mathscr{G}$ by dropping the orientations. The same holds also if A_k has not special neighbors, with the only exception of the non-special parent of A_k that is also parent of the special siblings of A_k.*

Proof of Lemma 15 According to Definition 3, the non-special nodes of $\mathscr{G}$ cannot receive incoming arcs. Thus, a non-special parent V of A_k is a leaf node of the undirected tree corresponding to $\mathscr{G}$ if and only if V has not any child in addition to A_k.

Let U be the node of $\mathscr{G}$ such that A_k is the s-node of $\mathscr{G}$ (or one of them, if there are many) at maximum distance from U.

If A_k has a special neighbor, it should be unique because of (i) in Lemma 14. Let A_l be this node. The undirected path from U to A_k should cross A_l, because otherwise A_l would be more distant from U than A_k. If a non-special parent of A_k would have a child, this node would be special by definition of s-network and would be an s-node more distant from U than A_k. This is against the definition of U. Thus, in this case, the non-special parents of A_k cannot have any child. That proves the first part of the Lemma.

If A_k has no special neighbors, it should have at least one special sibling because of Theorem 9. Point (ii) in Lemma 14 states that A_k and its special siblings have a single common parent, say S. The path from U to A_k crosses S, because otherwise the special siblings of A_k would be more distant from U, than A_k. Thus, the non-special parents of A_k different from S cannot have any child, because otherwise their child would be s-nodes more distant from U than A_k. That proves the second part of the Lemma. $\square$

Lemma 16 *Let $\mathscr{G}$ be an s-polytree with at least two s-nodes. Let A_k be a lonely node of $\mathscr{G}$ and U a node of $\mathscr{G}$ such that A_k is the s-node of $\mathscr{G}$ (or one of them, if there are many) at maximum distance from U. Then, U cannot be included in $\tilde{A}_k^+$.*

Proof of Lemma 16 Because of its definition, $\tilde{A}_k^+$ cannot include U, if $\delta(U, A_k) > 1$ and also if U is a child of A_k.

If U is a parent of A_k and it is also special, U should appear both in A_k^+ and U^+. Therefore, U cannot be included in $\tilde{A}_k^+$.

If U is a non-special parent of A_k, let A_l be a second s-node of $\mathscr{G}$. A_l should be a neighbor of U, because otherwise it would be more distant from U than A_k. According to Definition 3, A_l cannot be a parent of the non-special node U. Thus, A_l is a child of U. This means that U appears both in A_k^+ and A_l^+, and therefore cannot be in $\tilde{A}_k^+$.

Finally, it is obvious to see that, U cannot coincide with A_k, because otherwise the remaining s-nodes of $\mathscr{G}$ would be more distant from U than $A_k = U$. $\square$

Lemma 17 *A_k is the only special node that can appear in $\tilde{A}_k^+$.*

Proof of Lemmma 17 A_k is clearly the only special node included in A_k^+ if A_k has no special neighbors. Thus, in this case, A_k is the only s-node that can be in $\tilde{A}_k^+$. If A_k has some special neighbor, then this is unique because of (i) in Lemma 14. Let A_l be this s-node. If A_k is a parent of A_l, A_k appears both in A_k^+ and A_l^+. This means that A_k cannot be in $\tilde{A}_k^+$. Thus, $\tilde{A}_k^+$ includes only the parents of A_k and none of them is special, because A_l is the only special neighbor of A_k. In this case, therefore, no s-nodes are in $\tilde{A}_k^+$.

Finally, if A_l is parent of A_k, all the parents of A_k different from A_l are non-special, because A_l is the only special neighbor of A_k. Thus, A_k and A_l are the only s-nodes of A_k^+. But A_l appears also in A_l^+ and cannot be in $\tilde{A}_k^+$. Thus, A_k is the only s-node that can appear in $\tilde{A}_k^+$. $\square$

Proof of Theorem 10 The nodes of $\tilde{A}_k^+$, removed from $\mathscr{G}$ to obtain $\mathscr{G}'$, appear only in the potential ϕ_k by definition of $\tilde{A}_k^+$. All the potentials associated to the s-nodes of $\mathscr{G}'$ are therefore well defined.

Let S be the variable of A_k^+ not included in $\tilde{A}_k^+$. If $S = A_k$, then $\tilde{A}_k^+$ includes all the parents of A_k, while, if $S \in \Pi_{A_k}$, $\tilde{A}_k^+$ should include A_k. In the first case, to obtain $\mathscr{G}'$, we remove from $\mathscr{G}$ all the arcs having A_k as second endpoint, while in the second case A_k itself is removed. In any case, the condition about the second endpoints of the arcs of an s-network is always satisfied by $\mathscr{G}'$. That proves that $\mathscr{G}'$ is an s-network.

In order to prove that $\mathscr{G}'$ is an s-polytree, let U be the node of $\mathscr{G}$ such that A_k is the s-node of $\mathscr{G}$ (or one of them, if there are many) at maximum distance from U.

If A_k actually has a special neighbor, say A_l, we distinguish whether A_k is a parent or a child of A_l.

If A_k is a parent of A_l, then A_k appears both in A_k^+ and A_l^+ and $\tilde{A}_k^+ = \Pi_{A_k}$. According to (i) in Lemma 14, A_l is the only special neighbor of A_k and all the parents of A_k should be non-special.

Therefore, to obtain $\mathscr{G}'$ from $\mathscr{G}$, we remove the non-special parents of the lonely node A_k. According to Lemma 15, these nodes are leaf nodes in the tree corresponding to $\mathscr{G}$. Thus, $\mathscr{G}'$ is a polytree.

If A_l is a parent of A_k, A_l appears both in A_k^+ and A_l^+. This means that $\tilde{A}_k^+$ is composed by A_k and the parents of A_k different from A_l.

The parents of A_k different from A_l cannot be special because of (i) in Lemma 14. These nodes are therefore non-special parents of a lonely node and they should be leaf nodes in the undirected tree corresponding to $\mathscr{G}$ because of Lemma 15.

Furthermore, A_k cannot have any child. The path from U to A_k crosses A_l because otherwise A_l would be more distant from U than A_k. If A_k would have a child, this node would be special because of Definition 3, resulting an s-node more distant from U than A_k.

To obtain $\mathscr{G}'$ from $\mathscr{G}$, we can therefore remove first the parents of A_k different from A_l. Once we have removed these leaf nodes, A_k has a single parent (namely A_l) and no children. Thus, removing also A_k, we obtain a polytree, that is $\mathscr{G}'$.

If A_k has not special neighbors, it should have at least a special sibling because of Theorem 9. All the special siblings of A_k have a single parent in common with A_k because of (ii) in Lemma 14. Let S be this non-special parent of A_k. $\tilde{A}_k^+$ includes A_k and the parents of A_k different from S.

According to Lemma 15, the parents of A_k different from S are leaf nodes in the undirected tree corresponding to $\mathscr{G}$.

A_k cannot have any child also in this case. The path from U to A_k crosses S because otherwise the special siblings of A_k would be more distant from U than A_k. If A_k would have a child, this node would be special because of Definition 3, resulting an s-node more distant from U than A_k.

It is therefore possible to obtain $\mathscr{G}'$ from $\mathscr{G}$, removing first the parents of A_k different from S. Once we have removed these leaf nodes, A_k has a single parent (namely S) and no children. Thus, removing also A_k, we obtain a polytree, that is exactly $\mathscr{G}'$. $\square$

Proof of Theorem 11 U is not included in $\tilde{A}_k^+$ because of Lemma 16 and therefore it should be a node of $\mathscr{G}'$. Furthermore, all the s-nodes of $\mathscr{G}$ different from A_k are s-nodes of $\mathscr{G}'$ because of Lemma 17. Thus, $\mathscr{G}'$ includes U and all the s-nodes of $\mathscr{G}'$ except A_k. The removal of some arcs and some nodes from $\mathscr{G}$ to obtain $\mathscr{G}'$, which is connected because of Theorem 10, cannot modify the distances between U and the s-nodes different from A_k. That means that the $A_{k'}$ is the s-node of $\mathscr{G}'$ at maximum distance from U. $\square$

Proof of Theorem 12 The subroutine *findDistances* is known to be linear in the number of arcs of $\mathscr{G}$ [12]. On the other hand, *findConjugate*(A_k,$\mathscr{G}$) requires a number of operations equal to the number of neighbors of A_k. The children of A_k should be s-nodes because of Definition 3, while A_k has at most a special neighbor, and hence a children, because of (i) in Lemma 14. That means that A_k has at most a children. The number of neighbors of A_k is therefore dominated by $|\Pi_{A_k}| + 1$. The subroutine *findConjugate* is invoked m times, and the overall number of operations can therefore be bounded by $\sum_{i=0}^{m} |\Pi_{A_k}| + m$. But the first term of this sum represents the number of arcs of $\mathscr{G}$ because of Definition 3. Thus, also this part of the algorithm takes only a linear number of operations. Finally the evaluation of $\phi_k^{-\tilde{A}_k^+}$ was already noted to take place in a domain of the same dimension of those defined in input. That proves the thesis. $\square$

References

[1] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann, San Mateo, 1988.

[2] R. J. A. Little, D. B. Rubin, Statistical Analysis with Missing Data, Wiley, New York, 1987.

[3] P. Grunwald, J. Halpern, Updating probabilities, Journal of Artificial Intelligence Research 19 (2003) 243–278.

[4] G. de Cooman, M. Zaffalon, Updating beliefs with incomplete observations, Artificial Intelligence 159 (2004) 75–125.

[5] M. Zaffalon, The naive credal classifier, Journal of Statistical Planning and Inference 105 (1) (2002) 5–21.

[6] G. F. Cooper, The computational complexity of probabilistic inference using Bayesian belief networks, Artificial Intelligence 42 (1990) 393–405.

[7] P. P. Shenoy, J. Kohlas, Computation in valuation algebras, in: J. Kohlas, S. Moral (Eds.), Handbook of Defeasible Reasoning and Uncertainty Management Systems, Vol. 5, Kluwer, Dordrecht, 2000, pp. 5–39.

[8] C. F. Manski, Partial Identification of Probability Distributions, Springer-Verlag, New York, 2003.

[9] P. Walley, Statistical Reasoning with Imprecise Probabilities, Chapman and Hall, New York, 1991.

[10] C. Papadimitriou, Computational Complexity, Addison-Wesley, 1994.

[11] M. R. Garey, D. S. Johnson, Computers and Intractability; a Guide to the Theory of NP-completeness, Freeman, San Francisco, 1979.

[12] S. Even, Graph Algorithms, Computer Science Press, California, 1979.

[13] A. Antonucci, A. Salvetti, M. Zaffalon, Assessing debris flow hazard by credal nets, in: M. Lopez-Diaz, M. A. Gil, P. Grzegorzewski, O. Hryniewicz, J. Lawry (Eds.), SMPS 2004, Proceedings of the Second International Conference on Soft Methods in Probability and Statistics, Springer, 2004, pp. 125–132.

[14] M. Zaffalon, Conservative rules for predictive inference with incomplete data, in: F. G. Cozman, R. Nau, T. Seidenfeld (Eds.), ISIPTA '05, Proceedings of the Fourth International Symposium on Imprecise Probabilities and Their Applications, SIPTA, 2005, pp. 406–415.

[15] F. G. Cozman, Credal networks, Artificial Intelligence 120 (2000) 199–233.

[16] A. Antonucci, M. Zaffalon, Equivalence between Bayesian and credal nets on an updating problem, in: SMPS 2006, Proceedings of the Third International Conference on Soft Methods in Probability and Statistics, 2006, to appear.