Fast Algorithms for Robust Classification with Bayesian Nets

Alessandro Antonucci
IDSIA
Galleria 2
CH-6928 Manno (Lugano)
Switzerland
alessandro@idsia.ch

Marco Zaffalon
IDSIA
Galleria 2
CH-6928 Manno (Lugano)
Switzerland
zaffalon@idsia.ch

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.

Keywords. Bayesian networks, missing data, conservative updating rule, credal classification.

1 Introduction

Probabilistic expert systems yield conclusions on the basis of *evidence* about a domain. For example, systems based on *Bayesian networks* [11] (see Section 2.1) are queried for updating the confidence on a target variable given 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 [8], which is often unrealistic [7].

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. 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* [13] (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 is 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 imposed 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 [1]; 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*.

[1] The set of nodes made by its parents, its children, and the parents of the children.

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. Our algorithm has indeed quadratic time complexity also in many cases when the class variable with its Markov blanket form a multiply connected graph. 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. We achieve this goal, which is relatively involved from the technical point of view, in two steps. We firstly introduce a new kind of network model, called *s-network*, and show in Section 4 how to make efficient computations on s-networks whose graph is made by a set of polytrees (or *polyforest*). Secondly, we show in Section 5 that a problem of classification with CUR on Bayesian networks can be transformed into an equivalent problem on s-networks, so that if the latter is a polyforest, the original problem is solved efficiently.

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. Comments on the results are reported in the Conclusions (Section 6). A numerical example is reported in Appendix A.

2 Setup

2.1 Bayesian Networks

Consider the random variables $A_0, \ldots, A_n$. The 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 [11] 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 [8]. Unfortunately, MAR is quite a strong assumption [7] and for this reason MAR-based approaches are somewhat criticized (see also [9]).

Following a deliberately conservative approach, de Cooman and Zaffalon [4] have instead used *coherent lower previsions* [12] 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:

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

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

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 $c' {>} c''$, 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*[2] proposed by Walley [12, 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 [13].

It is easy to show that testing whether $c' {>} c''$ 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[3] 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.

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 [1], 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 $\mathscr{Q}$ [6, p. 38]. In our case, we will transform a well known NP-complete problem, called 3-satisfiability (3SAT) [6], 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 $\mathscr{K} = \{K_1, \ldots, K_m\}$ be a non-empty collection of *clauses*, which are disjunctions of triples of literals, corresponding to different[4] 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.

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

Theorem 1. *CCURD is coNP-complete.*

Proof. 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

$$\mathscr{E} = \{(U_{\alpha_{ij}}, K_j) \mid \begin{smallmatrix} i = 1,2,3, \\ j = 1,\ldots,m \end{smallmatrix}\} \cup \{(C, K_j) \mid j = 1, \ldots, m\}, \tag{5}$$

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 $\mathscr{U}$.

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 de-

[2] Strict preference can be applied also when the space of options is randomized. We do not investigate such a case here because we ideally work in a classification setup, where randomization is not used very frequently. This nevertheless, it could be worth considering the extension to the mentioned case for other kinds of applications.

[3] 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.

[4] This assumption is not included in the original transformation of the prototypical NP-complete problem SAT to 3SAT. Nevertheless, the transformation (see for example [6, p. 48]) does not require any clause to include literals corresponding to the same variable. Thus, also this version of 3SAT is NP-complete.

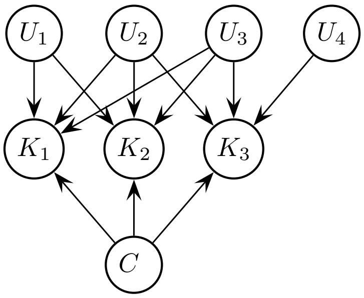

Figure 1: A Bayesian net corresponding to a 3SAT instance with $\mathscr{U} = \{U_1, U_2, U_3, U_4\}$ and $\mathscr{K} = \{(u_1 \vee u_2 \vee u_3), (\neg u_1 \vee \neg u_2 \vee u_3), (u_2 \vee \neg u_3 \vee u_4)\}$.

fine 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	$u_{\alpha_{1j}} \vee u_{\alpha_{2j}} \vee u_{\alpha_{3j}}$	$p(K_j = \mathrm{T} \vert c, u_{\alpha_{1j}}, u_{\alpha_{2j}}, u_{\alpha_{3j}})$
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 the K_j.

The directed acyclic graph $\mathscr{G}$, together with the specified mass functions, define 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:

$$\min_{\substack{u_j \in \{F,T\}, \\ U_j \in \mathscr{U}}} \prod_{i=1}^{m} f_i(u_{\alpha_{1i}}, u_{\alpha_{2i}}, u_{\alpha_{3i}}), \tag{6}$$

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

$$f_i(u_{\alpha_{1i}}, u_{\alpha_{2i}}, u_{\alpha_{3i}}) := \frac{p(K_i = \mathrm{T} \vert c', u_{\alpha_{1i}}, u_{\alpha_{2i}}, u_{\alpha_{3i}})}{p(K_i = \mathrm{T} \vert c'', u_{\alpha_{1i}}, u_{\alpha_{2i}}, u_{\alpha_{3i}})}. \tag{7}$$

Using the values of Table 1, the functions in (7) take the form:

$$f_i(u_{\alpha_{1i}}, u_{\alpha_{2i}}, u_{\alpha_{3i}}) = \begin{cases} 2^{-1} & \text{if } u_{\alpha_{1i}} \vee u_{\alpha_{2i}} \vee u_{\alpha_{3i}} = \mathrm{T} \\ 2^{m} & \text{otherwise.} \end{cases} \tag{8}$$

According to (8), 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 (7) in (6) are simultaneously minimized. The minimum (6) 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 (6) 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 (8) 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. Let CCURD' be the complement of CCURD. In order to prove the hardness of CCUR we consider a polynomial-time Turing reduction [6, 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 Theory

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 quadratic time algorithm that works in many cases also when B^+ is multiply connected.

The development of the new algorithm passes through the definition of a new kind of graphical model, called *s-network*, which allows us to abstract the main components of a CCUR problem. This is done in Section 4.1, which also defines the minimum value of an s-network. Section 4.2 develops an algorithm to compute such a minimum when the graph associated with the s-network is a polyforest. Finally, we show in Section 5 how to transform a CCUR problem to the problem of computing the minimum of an s-network. In this way we expand the class of efficiently solvable CCUR problems to those in which B^+, after the transformation, becomes a polyforest.

4.1 Basic Definitions

Let us first introduce the following:

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 function $f_{A_i}(A_i^+)$, 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 functions f_{A_i} is called* s-network.

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

$$\min_{\substack{a_j^+ \in \mathscr{A}_j^+,\\ j \in \{0,\ldots,m\}}} \prod_{i=0}^{m} f_{A_i}(a_i^+). \tag{9}$$

Note that Definition 3 does not exclude the case of disconnected s-networks. If a connected component $\mathscr{G}_i$ of a (disconnected) s-network $\mathscr{G}$ includes at least one s-node, we can regard $\mathscr{G}_i$, together with the functions of $\mathscr{G}$ corresponding to the s-nodes of $\mathscr{G}_i$, 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 focus on the task of computing minima of s-networks. According to Theorem 4, we can consider only the case of a connected s-network with at least one s-node.

4.2 Fast Computation of Minima of S-networks

We call *s-polytree* an s-network $\mathscr{G}$ such that the underlying graph is a 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 $d(U, V)$ be the number of edges making up this path. The map d is clearly a metric over $\mathscr{V}$ and $d(U, V)$ 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). The lonely nodes of an s-polytree can be characterized by the following:

Theorem 5. *Let $\mathscr{G}$ be an s-polytree and A_k a lonely node of $\mathscr{G}$. The variables in A_k^+, with the possible exception of a variable called S, appear only in the argument of f_{A_k}.*

Given the lonely node A_k, let us denote by $\tilde{A}_k^+$ the vector variable that includes all the variables in A_k^+ except S. Theorem 5 states that the variables in $\tilde{A}_k^+$ appear only in the argument of f_{A_k}, while no definite information is given about S. A further characterization of $\tilde{A}_k^+$ comes from the following:

Theorem 6. *A_k is the only special node that can appear in $\tilde{A}_k^+$.*

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^+. Furthermore, we call *siblings* two distinct children of the same parent. The conjugate nodes of a lonely node are characterized by the following:

Theorem 7. *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 special 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 special sibling.*

According to Theorem 7, for each lonely node A_k of an s-polytree with at least two s-nodes, there is at least a conjugate A_l and the variable $S \in A_k^+$, which does not appear in $\tilde{A}_k^+$, is in the argument of f_{A_l}.

Once we have detected a lonely node and a corresponding conjugate, it is possible to construct a second s-polytree, with an s-node less, and the same minimum. The procedure is reported in the following:

Theorem 8. *Let $\mathscr{G}$ be an s-polytree with at least two special nodes. Let A_k be a lonely node of $\mathscr{G}$ and A_l a conjugate of A_k. The minimum of $\mathscr{G}$ coincides with that of a second s-polytree $\mathscr{G}'$, obtained marking A_k as not special, re-defining $f_{A_l}(a_l^+)$ for all a_l^+ as*

$$f_{A_l}(a_l^+) \cdot \min_{\tilde{a}_k^+} f_{A_k}(a_k^+), \tag{10}$$

and removing all the nodes in $\tilde{A}_k^+$ from $\mathscr{G}$.

Algorithm 1 represents an obvious implementation of the reduction from $\mathscr{G}$ to $\mathscr{G}'$ given by Theorem 8.

```
mark $A_k$ as not special;
FOREACH $V \in A_k^+$ {
        IF $V \notin A_l^+$ {
                put $V$ in $\tilde{A}_k^+$; }}
FOREACH $a_l^+ \in \mathscr{A}_l^+$ {
        $f_{A_l}(a_l^+) = f_{A_l}(a_l^+) \cdot \min_{\tilde{a}_k^+} f_{A_k}(a_k^+)$; }
FOREACH $V \in \tilde{A}_k^+$ {
        remove $V$ from $\mathscr{G}$; }
RETURN $\mathscr{G}$;
```

Algorithm 1: The `reduce` function. The inputs are an s-polytree $\mathscr{G}$ and the s-nodes A_k and A_l. The output `reduce`($\mathscr{G}$,A_k,A_l) is the s-polytree $\mathscr{G}'$ with a special node less as in Theorem 8.

To actually apply this reduction, a procedure to detect lonely nodes and the corresponding conjugates in s-polytrees is required.

Given an arbitrary node U of an s-polytree $\mathscr{G}$, let us evaluate the distances $d(U, A_j)$ $(j = 0, \ldots, m)$. By definition, the node A_k with $k := \arg\max_{j=0,\ldots,m} d(U, A_j)$ is a lonely node of $\mathscr{G}$.

Let `distances`($\mathscr{G}$,U) be the procedure returning the distances between U and the s-nodes of $\mathscr{G}$. The well known *depth first search* (DFS) algorithm [5] over the undirected graph obtained forgetting the orientation of the arcs of $\mathscr{G}$ with starting node U can be used to implement the procedure. The computational complexity of the algorithm is known to be linear in the number of arcs of $\mathscr{G}$ [5].

Regarding the detection of a conjugate node given its lonely node, Theorem 7 suggests an obvious procedure reported by Algorithm 2.

Given an s-polytree $\mathscr{G}$, we are therefore able to find a lonely node and a node conjugate of it. Afterwards, we invoke Algorithm 1 to produce a second s-polytree $\mathscr{G}'$ with an s-node less and the same minimum.

According to Theorem 8, the output of this reduction is still an s-polytree. It is therefore possible to iterate this procedure, until an s-polytree with a single s-node is returned.

The following theorem makes it easier the detection of a lonely node of $\mathscr{G}'$.

```
1. FOREACH $V \in$ neighbors($A_k$) {
2.         IF $V$ is special {
3.                 $A_l$ := $V$;
4.                 GO TO 8; }}
5. FOREACH $V \in$ siblings($A_k$) {
6.         IF $V$ is special {
7.                 $A_l$ := $V$; }}
8. RETURN $A_l$;
```

Algorithm 2: 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 obvious subroutines `neighbors` and `siblings` return respectively the neighbors and the siblings of the node in their argument.

Theorem 9. *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-node at maximum distance from U (or one of them, if there are many). Let A_l be a conjugate of A_k, that is a lonely node of $\mathscr{G}$ by definition. Thus, $A_{k'}$ is a lonely node of $\mathscr{G}' = \texttt{reduce}(\mathscr{G}, A_k, A_l)$.*

Algorithm 3 reports the whole iterative procedure to calculate the minimum of an s-network.

```
$U$ := randomly chosen node of $\mathscr{G}$;
$(d_1, \ldots, d_m)$ := distances($\mathscr{G}$,$U$);
WHILE number of s-nodes in $\mathscr{G} > 1$ {
        $k$ := $\arg\max d_j$;
        $A_l$ := findConjugate($A_k$,$\mathscr{G}$);
        $\mathscr{G}$ := reduce($\mathscr{G}$,$A_k$,$A_l$);
        remove $d_k$, from $(d_1, \ldots, d_m)$;}
RETURN $\min_{a_l^+} f_{A_l}(a_l^+)$;
```

Algorithm 3: The pseudo-code of the full minimization routine. In input we have an s-polytree $\mathscr{G}$. The output is the minimum of the s-polytree.

Concerning the computational complexity of Algorithm 3, it is obvious to check that the subroutines `reduce` and `findConjugate` are linear in the number of nodes of $\mathscr{G}$, while `distances` was already noted to be linear. The latter is invoked only once, while the former two are invoked as many times as many s-nodes minus one are in the s-network. The running time of the full algorithm is therefore at most quadratic in the input size.

Finally, the algorithm works only if the graph underlying the s-network is a polytree. Thus, if $\mathscr{G}(\mathscr{V}, \mathscr{E})$ is this graph, the condition $|\mathscr{V}| = |\mathscr{E}| + 1$ can be used as an obvious applicability check.

Remember that we are focusing on connected s-networks. In the general case of a disconnected s-network $\mathscr{G}$, we have only to to check whether or not the graph is a polyforest. In the positive case, the algorithm in Table 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.

5 Efficient CUR-based Classification

5.1 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.

```
$\mathscr{G}_I$ := $B^+$;
FOREACH $V \in \mathscr{V}$ {
  IF $V$ = $C$ OR $C$ parent of $V$ {
    mark $V$ as special; }}
FOREACH $\epsilon \in \mathscr{E}$ {
  IF $\mathrm{T}(\epsilon) \in E$ OR $T(\epsilon)$ = $C$ {
    remove $\epsilon$; }}
FOREACH $V \in E$ {
  IF $V$ not special {
    remove $V$; }}
```

Algorithm 4: 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.

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:

$$f_{A_0}(a_0^+) \quad := \quad \frac{p(c'|\pi_C)}{p(c''|\pi_C)} \tag{11}$$

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

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

Theorem 10. *I is true if and only if the minimum of the s-network $\mathscr{G}_I$ is greater than one.*

Therefore, Algorithm 3 can solve a CCURD instance I, such that the corresponding s-network $\mathscr{G}_I$ is polyforest-shaped. Finally, it is easy to check that the transformation from I to the s-network $\mathscr{G}_I$ is linear in the size of I.

5.2 Solving CCUR Problems

Theorem 10 is the basis to solve efficiently 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 $c_i > c_j$, while the second checks if $c_j > c_i$. Whenever one of these CCURD instances is true, the dominated class is rejected. Algorithm 5 reports the full procedure detecting the optimal classes.

Concerning the computational complexity of Algorithm 5, the total number of solved CCURD instances is quadratic in the input size, being exactly $r \cdot (r - 1)$.

Finally, to detect whether or not this approach can be used to solve a given CCUR instance I, it is sufficient to check if the graph $\mathscr{G}_I$ returned by Algorithm 4 is a polyforest. The algorithm obtains $\mathscr{G}_I$ removing some nodes and arcs from B^+. Therefore $\mathscr{G}_I$ can be a polyforest also if the original Markov blanket is multiply connected (e.g. the Bayesian network reported in App. A).

In analogy with [4, Section 6], the common technique called *loop cutset conditioning* can be used to solve a CCUR instance I, when $\mathscr{G}_I$ is not a polyforest. In this case the computation will take exponential time.

```
$\mathscr{C}_{opt}$ := $\mathscr{C}$;
FOR $i = 1, \ldots, r$ {
    FOR $j = 1, \ldots, r$ {
        IF $i < j$ {
            IF $c_i \succ c_j$ {
                remove $c_j$ from $\mathscr{C}_{opt}$;}
            IF $c_j \succ c_i$ {
                remove $c_i$ from $\mathscr{C}_{opt}$;}}}}
RETURN $\mathscr{C}_{opt}$;
```

Algorithm 5: 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 $\mathscr{C}_{opt}$.

6 Conclusions

Probabilistic expert systems suggest actions on the basis of the available evidence about a domain. Often such an evidence is only partial in many real applications, 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 state of near-ignorance about the missingness process, and by updating beliefs accordingly. In order to make the rule profitably used in practice 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 traditional 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 form a multiply connected net. 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, for 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, it seems possible to proceed as in [4, Sect. 7] to employ our algorithm also in the case of *credal networks* [3], which are graphical models extending the formalism of Bayesian networks by allowing sets of mass functions.

Finally, there appear to be strong connections between the algorithms proposed here and algorithms based on *junctions trees*, in particular *max-marginalization* [2]. These connections could be deepened in future work to represent the problem in a more standard setup, and perhaps exploited to achieve a more general or efficient formulation.

Acknowledgements

This research was partially supported by the Swiss NSF grant 2100-067961.

A A Numerical Example

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.

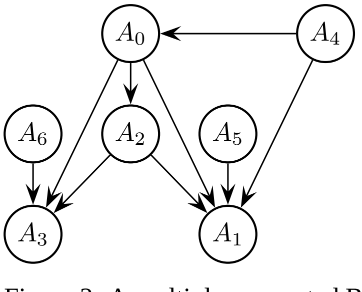

Figure 2: A multiply connected Bayesian network.

We assume uniform unconditional mass functions for the root nodes, while Tables 6, 7, 8 and 9 specify the conditional mass functions for the remaining nodes.

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

Table 6: Conditional mass functions for node C.

c	a_2	a_4	a_5	$p(A_1=\mathrm{T}\,\|c,a_2,a_4,a_5)$
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 7: Conditional mass functions for node A_1.

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

Table 8: Conditional mass functions for node A_2.

The decision whether $c' > c''$ 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 4 to construct the graph $\mathscr{G}_I$ corresponding to the instance I. The result is reported in Figure 3 and can be easily recognized to be a polytree.

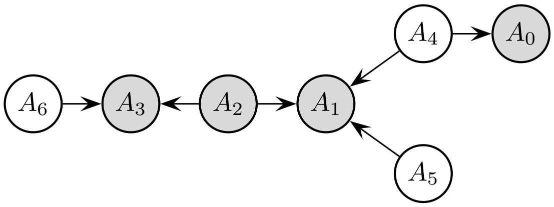

Figure 3: The polytree obtained applying Algorithm 4 to the Bayesian network in Figure 2. The s-nodes are displayed in gray.

c	a_2	a_6	$p(A_3=\mathrm{T}\,\|c,a_2,a_6)$
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 9: Conditional mass functions for node A_3.

According to the procedure described in Section 5.1, 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 specifications in Tables 6–9 to define a function for each special node of $\mathscr{G}_I$ as in (11) and (12). $\mathscr{G}_I$ together with this set of functions is an s-polytree and Algorithm 3 can therefore be used to compute its minimum.

Let $U := A_6$ be the randomly chosen node. The distances between U and the s-nodes of $\mathscr{G}_I$ are: $d_0 = 5$, $d_1 = 3$, $d_2 = 2$, $d_3 = 1$. Thus, A_0 is a lonely node of $\mathscr{G}_I$, while its only conjugate node is the special sibling A_1. Clearly, in this case, $\tilde{A}_0^+ = A_0$, which is a constant, and (10) becomes

$$f_{A_1}(a_1,a_2,a_4,a_5)\cdot f_{A_0}(a_4). \tag{13}$$

We finally obtain $\mathscr{G}_I'$, removing A_0 from $\mathscr{G}_I$. A_1 is a lonely node of $\mathscr{G}_I'$ with conjugate A_2 and $\tilde{A}_1^+ = (A_1,A_4,A_5)$. Thus, (10) takes the form

$$f_{A_2}(a_2)\cdot\min_{a_1,a_4,a_5} f_{A_1}(a_1,a_2,a_4,a_5). \tag{14}$$

$\mathscr{G}_I''$ is indeed obtained removing A_1, A_4 and A_5. A_2 is a lonely node of this s-polytree with conjugate A_3, and $\tilde{A}_2^+$ is empty. The re-definition of the function associated to the conjugate node is therefore simply

$$f_{A_3}(a_3,a_2,a_6)\cdot f_{A_2}(a_2). \tag{15}$$

The final mark of A_2 as non-special node leads to an s-polytree with a single s-node, whose minimum coincides with the minimum of $\mathscr{G}_I$, being exactly

$$\min_{a_3,a_2,a_6} f_{A_3}(a_3,a_2,a_6) = 0.76 \tag{16}$$

According to Theorem 10, I is therefore false and c' does not dominate c''.

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 0.02. $\overline{I}$ 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.

In contrast, if we assume that $A_6 =$T and the remaining attribute variables are missing, we find, with similar calculations, the numerical value 1.19 for the minimum of $\mathscr{G}_I$ and 0.02 for $\mathscr{G}_{\overline{I}}$. In other words, c' dominates c'' and is therefore the only optimal class.

References

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

[2] R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer-Verlag, New York, 1999.

[3] F. G. Cozman. Credal networks. *Artificial Intelligence*, 120:199–233, 2000.

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

[5] S. Even. *Graph Algorithms*. Computer Science Press, 1979.

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

[7] P. Grunwald and J. Halpern. Updating probabilities. *Journal of Artificial Intelligence Research*, 19:243–278, 2003.

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

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

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

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

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

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