International Journal of Approximate Reasoning 52 (2011) 2–18

Contents lists available at ScienceDirect

International Journal of Approximate Reasoning

journal homepage: www.elsevier.com/locate/ijar

Acyclic directed graphs representing independence models

Marco Baioletti [a], Giuseppe Busanello [b], Barbara Vantaggi [b,*]

[a] Dip. Matematica e Informatica, Università di Perugia, Italy
[b] Dip. Scienze di Base e Applicate per l'Ingegneria, Università, La Sapienza, Roma, Italy

ARTICLE INFO

Article history:
Available online 15 October 2010

Keywords:
Independence models
Graphoid properties
Inferential rules
Acyclic directed graphs
Perfect map

ABSTRACT

In this paper we study the problem of representing probabilistic independence models, in particular those closed under graphoid properties. We focus on acyclic directed graph (DAG): a new algorithm to build a DAG, given an ordering among random variables, is described and peculiarities and advantages of this approach are discussed. Moreover, we provide a necessary and sufficient condition for the existence of a perfect map representing an independence model and we describe an algorithm based on this characterization.

1. Introduction

Graphical models [7,8,11–13,17,21,26,31,34–39,44] play a fundamental role in probability and multivariate statistics, as well as in artificial intelligence, since they are used as a tool for representing conditional independence models. The usefulness of graphical models is not limited to the probabilistic setting, indeed they have been extended also to other frameworks such as possibility, belief function, credal set and lower probability [1,5,6,9,10,14,20,22,27,40,42].

An important problem is to find for a given independence model J all the independence statements θ which are implied by J, i.e. which hold under any probability distributions agreeing with J. This problem, called implication problem, has not been solved yet and perhaps is undecidable. However it is possible to study its syntactical counterpart, i.e. to find if θ can be derived from J using some axiomatic system.

In particular, we deal with conditional independence models which are closed with respect to the graphoid properties. These models can be generated by a strictly positive probability under the classical notion of independence [12].

It is then interesting to find the closure $\bar{J}$ of J with respect to graphoid properties, but since this set can be exponentially larger than J, a suitable set J_* ("fast closure") of $\bar{J}$, gathering the same information as $\bar{J}$, has been introduced [34,35,2].

Another relevant problem is to represent a set J of conditional independence relations by means of an acyclic directed graph (DAG). We recall that a DAG gives a very compact and human-readable representation, unfortunately it is known that there are sets of independencies which admit no perfect maps. The problem of the existence of a perfect map has been studied by many authors by providing partial answers in terms of necessary conditions (see for instance [26]), or complete solutions [30], which require to solve a large number of implication/deduction problems.

In [3] we introduced a sufficient condition for the existence of a perfect map in terms of existence of a certain ordering among the random variables, and we described the procedure BN-DRAW which builds the corresponding independence map given an ordering. This condition, as well as BN-DRAW, uses the "fast closure" [2].

For semi-graphoid structures a construction, similar to the fast closure, is given in [34] and it is used in [15] to introduce a necessary condition for the existence of a perfect map.

* Corresponding author.
E-mail addresses: baioletti@dipmat.unipg.it (M. Baioletti), busanello@dmmm.uniroma1.it (G. Busanello), vantaggi@dmmm.uniroma1.it (B. Vantaggi).

0888-613X/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.ijar.2010.09.005

In [3] we also defined a correct, but incomplete, algorithm to find a perfect map. A backtracking-based procedure searches for a suitable ordering which satisfies the sufficient condition. If such an ordering exists, a perfect map for the independence model can be found by using the procedure BN-DRAW. Since the above condition is not necessary, but only sufficient, as shown in Example 3 and in Example 4, it can fail even if a perfect map exists.

The main result of this paper is to provide a necessary and sufficient for the existence of a perfect map, which characterizes the orderings on which BN-DRAW extracts a perfect map, starting from the fast closure. These conditions rely on some constraints among the triples of the set J_* and their components.

The paper is structured as follows. In Section 2 we recall the concept of graphoid and the generalized rules, as introduced in [2]. In Section 3 we give some experimental results on the computation of the fast closure. In Sections 4 and 5 we recall some useful notions of graphs. Section 6 is devoted to discuss and to prove the characterization of DAG representability. In Section 7 an algorithm based on the previous result is described and some empirical results are given. Sections 8 finally describes some connections with the related literature and draws some conclusions.

2. Graphoid

Let $\widetilde{S} = \{Y_1, \ldots, Y_n\}$ be a finite not empty set of variables and $S = \{1, \ldots, n\}$ the set of indices associated to $\widetilde{S}$.

Given a (coherent) probability P on $\widetilde{S}$, a conditional independence statement $Y_A \perp\!\!\!\perp Y_B | Y_C$, compatible with P, is simply denoted by the ordered triple (A,B,C), where A, B, C are disjoint subsets of S. Then, in the following we do not distinguish $\widetilde{S}$ from S.

Furthermore, $S^{(3)}$ is the set of all (ordered) triples (A,B,C) of disjoint subsets of S, such that A and B are not empty. A conditional independence model $\mathcal{I}$ is a subset of $S^{(3)}$. In particular, as claimed in the introduction, we refer to a graphoid structure, which is a couple $(S,\mathcal{I})$, where $\mathcal{I}$ is a ternary relation on S satisfying the following properties (where A,B,C,D are pairwise disjoint subsets of S):

G1 if $(A,B,C) \in \mathcal{I}$, then $(B,A,C) \in \mathcal{I}$ (Symmetry);
G2 if $(A,B \cup C,D) \in \mathcal{I}$, then $(A,B,D) \in \mathcal{I}$ Decomposition);
G3 if $(A,B \cup C,D) \in \mathcal{I}$, then $(A,B,C \cup D) \in \mathcal{I}$ (Weak Union);
G4 if $(A,B,C \cup D) \in \mathcal{I}$ and $(A,C,D) \in \mathcal{I}$, then $(A,B \cup C,D) \in \mathcal{I}$ (Contraction);
G5 if $(A,B,C \cup D) \in \mathcal{I}$ and $(A,C,B \cup D) \in \mathcal{I}$, then $(A,B \cup C,D) \in \mathcal{I}$ (Intersection).

A semi-graphoid is a couple $(S,\mathcal{I})$ where $\mathcal{I}$ satisfies only the properties G1–G4.

Given a triple $\theta = (A,B,C)$ we denote by θ^T the triple obtained by applying G1 to θ (called the transpose of θ), i.e. $\theta^T = (B,A,C)$.

2.1. Generalized inference rules and fast closure

Given a set J of conditional independence statements, compatible with a probability, a relevant problem is to find the closure of J with respect to G1–G5

$$\bar{J} = \left\{\theta \in S^{(3)} : \theta \text{ is obtained from } J \text{ by G1—G5}\right\}.$$

A related problem, called deduction, concerns to establish whether a triple $\theta \in S^{(3)}$ can be derived from J, see [4,45]. It is clear that the deduction problem can be easily solved once the closure has been computed. But, the computation of $\bar{J}$ is infeasible because its size can be exponentially larger than the size of J. In [2] we described how it is possible to compute a smaller set of triples having the same information as the closure. The same problem has been already faced successfully in [35], with particular attention to semi-graphoid structures.

We recall in the following subsections some definitions and properties (see for more details [2]), which are used in the rest of the paper.

2.1.1. Generalized inclusion

Given a pair of triples $\theta_1, \theta_2 \in S^{(3)}$, we say that θ_1 is *generalized-included* in θ_2 (briefly g-included), in symbol $\theta_1 \sqsubseteq \theta_2$, if θ_1 can be obtained from θ_2 by a finite number of applications of the unary rules G1, G2 and G3.

Proposition 1. *Given $\theta_1 = (A_1,B_1,C_1)$ and $\theta_2 = (A_2,B_2,C_2)$, then $\theta_1 \sqsubseteq \theta_2$ if and only if the following conditions hold:*

(i) $C_2 \subseteq C_1 \subseteq (A_2 \cup B_2 \cup C_2)$;
(ii) *either* $A_1 \subseteq A_2$ *and* $B_1 \subseteq B_2$, *or* $A_1 \subseteq B_2$ *and* $B_1 \subseteq A_2$.

Generalized inclusion is strictly related to the concept of dominance [34,35] where θ_1 is said to be dominated by θ_2 (in symbol $\theta_1 \sqsubseteq_a \theta_2$) if the condition (i) of Proposition 1 holds and, moreover, $A_1 \subseteq A_2$ and $B_1 \subseteq B_2$.

The definition of g-inclusion between triples can be extended to sets of triples:

Definition 1. Let H and J be subsets of $S^{(3)}$. J is a covering of H (in symbol $H \sqsubseteq J$) if and only if for any triple $\theta \in H$ there exists a triple $\theta' \in J$ such that $\theta \sqsubseteq \theta'$.

2.1.2. Generalized rules

By using the relation $\sqsubseteq$ it is possible to define a generalization of the binary rules G4 and G5.

Given $\theta_1 = (A_1, B_1, C_1)$ and $\theta_2 = (A_2, B_2, C_2)$ in $S^{(3)}$, let $\mathcal{X}_i = (A_i \cup B_i \cup C_i)$ for $i = 1, 2$. Now, let:

$$W_C(\theta_1, \theta_2) = \{\tau : \theta'_1, \theta'_2 \vdash_{G4} \tau, \quad \text{with} \quad \theta'_1 \sqsubseteq_a \theta_1, \theta'_2 \sqsubseteq_a \theta_2\},$$

where $\theta'_1, \theta'_2 \vdash_{G4} \tau$ means that τ is obtained by applying G4 to θ'_1 and θ'_2.

In [2] we proved that $W_C(\theta_1, \theta_2)$ is not empty if and only if the following conditions hold:

(1) $(A_1 \cap A_2) \neq \emptyset$;
(2) $C_1 \subseteq \mathcal{X}_2$ and $C_2 \subseteq \mathcal{X}_1$;
(3) $(B_1 \setminus C_2) \neq \emptyset$;
(4) $B_2 \cap \mathcal{X}_1 \neq \emptyset$;
(5) $|(B_1 \setminus C_2) \cup (B_2 \cap \mathcal{X}_1)| \geqslant 2$.

Moreover, if $W_C(\theta_1, \theta_2)$ is not empty, then the triple:

$$gc(\theta_1, \theta_2) = (A_1 \cap A_2, (B_1 \setminus C_2) \cup (B_2 \cap \mathcal{X}_1), C_2 \cup (A_2 \cap C_1)),$$

is in $W_C(\theta_1, \theta_2)$ and g-includes any triple belonging to $W_C(\theta_1, \theta_2)$.

By denoting with $GC(\theta_1, \theta_2)$ the set formed by the possible (i.e. belonging to $S^{(3)}$) triples among $gc(\theta_1, \theta_2)$, $gc(\theta_1, \theta_2^T)$, $gc(\theta_1^T, \theta_2)$ and $gc(\theta_1^T, \theta_2^T)$, it is possible to state a generalization of the inference rule G4:

G4* "generalized contraction" : from θ_1, θ_2 deduce any triple $\tau \in GC(\theta_1, \theta_2)$.

Note that G4* can be seen as a generalization of G4, which also uses G1–G3. In fact, if it is possible to apply G4 to θ_1, θ_2, obtaining τ, then $\tau = gc(\theta_1, \theta_2)$ and so τ is deducible from θ_1, θ_2 also through G4*. Otherwise, each triple deduced through G4* g-includes all the possible triples generated from θ_1, θ_2 through repeated applications of G1–G4.

A similar result, related to intersection property (see [2]) is given by considering the set:

$$W_I(\theta_1, \theta_2) = \{\tau : \theta'_1, \theta'_2 \vdash_{G5} \tau, \quad \text{with} \quad \theta'_1 \sqsubseteq_a \theta_1, \theta'_2 \sqsubseteq_a \theta_2\},$$

where $\theta'_1, \theta' \vdash_{G5} \tau$ means that τ is obtained by applying G5 to θ'_1 and θ'_2.

In particular, in [2] we proved that, given $\theta_1 = (A_1, B_1, C_1), \theta_2 = (A_2, B_2, C_2)$, the set $W_I(\theta_1, \theta_2)$ is not empty if and only if the following conditions hold:

(1) $A_1 \cap A_2 \neq \emptyset$;
(2) $C_1 \subseteq \mathcal{X}_2$ and $C_2 \subseteq \mathcal{X}_1$;
(3) $B_1 \cap \mathcal{X}_2 \neq \emptyset$;
(4) $B_2 \cap \mathcal{X}_1 \neq \emptyset$;
(5) $|(B_1 \cap \mathcal{X}_2) \cup (B_2 \cap \mathcal{X}_1)| \geqslant 2$.

Moreover, if $W_I(\theta_1, \theta_2)$ is not empty, then the triple:

$$gi(\theta_1, \theta_2) = (A_1 \cap A_2, (B_1 \cap \mathcal{X}_2) \cup (B_2 \cap \mathcal{X}_1), (C_1 \cap A_2) \cup (C_2 \cap A_1) \cup (C_2 \cap C_1))$$

is in $W_I(\theta_1, \theta_2)$ and g-includes any triple in $W_I(\theta_1, \theta_2)$.

By denoting with $GI(\theta_1, \theta_2)$ the set formed by the possible triples among $gi(\theta_1, \theta_2), gi(\theta_1, \theta_2^T), gi(\theta_1^T, \theta_2)$ and $gi(\theta_1^T, \theta_2^T)$, it is possible to state a generalization of the inference rule G5:

G5* "generalized intersection" : from θ_1, θ_2 deduce any triple $\tau \in GI(\theta_1, \theta_2)$.

Note that G5* generalizes G5 in the same sense that G4* generalizes G4.

2.1.3. Fast closure

It is possible to compute the closure of a set J of triples in $S^{(3)}$ with respect to the generalized rules G4* and G5*, i.e. the set:

$$J^* = \{\tau : J \vdash_G^* \tau\}, \tag{1}$$

where $J \vdash_G^* \tau$ means that τ is obtained by applying a finite number of times the rules G4* and G5*.

In [2] the relationship between the two closures J^* and $\bar{J}$ is studied, in particular, we proved that $J^* \subseteq \bar{J}$ and $\bar{J} \sqsubseteq J^*$.

Note that J^* is a subset of $\bar{J}$ and it has the same information of $\bar{J}$. However, J^* can contain some "redundant" triples, i.e. some triples which are g-included in some other ones. Therefore, the set J^* can be reduced by using the concept of "maximal"

(with respect to g-inclusion) triple: given a set J of triples, a triple τ is maximal in J if there is no $\bar{\tau} \in J$ with $\bar{\tau} \neq \tau, \tau^T$ such that $\tau \sqsubseteq \bar{\tau}$. Actually, such reduction requires to delete the triples obtained through G1–G3.

We denote with $J_{/\sqsubseteq}$ the subset of J composed only by its maximal triples, moreover in the case that τ and τ^T are both maximal triples in J, then only one of them, arbitrarily chosen, is kept in $J_{/\sqsubseteq}$. The function which computes $J_{/\sqsubseteq}$ from a set J is called FindMaximal.

By using $\bar{J}_{/\sqsubseteq}$ instead of $\bar{J}$ there is no loss of information, in fact (see [2]):

$$\bar{J} \sqsubseteq \bar{J}_{/\sqsubseteq}.$$

Then, given a set J of triples in $S^{(3)}$, we compute J^* (see Eq. (1)) and then we take only its maximal triples, i.e. $J^*_{/\sqsubseteq}$. The set $J^*_{/\sqsubseteq}$ is called "fast closure" and it is denoted, for simplicity, with J_*.

2.2. Unique rule and algorithm FC1

A faster way to compute the fast closure is to exploit the property that the fast closure $\{\theta_1, \theta_2\}_*$ of a given pair $\theta_1, \theta_2 \in S^{(3)}$ is composed by a maximum of 9 extra triples, no matter how many variables occur in θ_1 and θ_2, as proved in [2]. In particular, let $K(\theta_1, \theta_2)$ be the set:

$$\{\theta_1, \theta_2, \varphi(\theta_1, \theta_2), \varphi(\theta_1^T, \theta_2), \varphi(\theta_1, \theta_2^T), \varphi(\theta_1^T, \theta_2^T), \varphi(\theta_2, \theta_1), \varphi(\theta_2^T, \theta_1), \varphi(\theta_2, \theta_1^T), \varphi(\theta_2^T, \theta_1^T), \nu(\theta_1, \theta_2)\},$$

where

$$\varphi(\theta_1, \theta_2) = (A_1 \cap A_2, B_1 \cup (B_2 \cap \mathcal{X}_1), C_1 \setminus B_2),$$

and

$$\nu(\theta_1, \theta_2) = ((A_1 \cap B_2) \cup (A_2 \cap B_1), (A_1 \cap A_2) \cup (B_1 \cap B_2), C_1 \cup C_2).$$

Note that:

$$\varphi(\theta_1, \theta_2) = gc[gc(\theta_1, \theta_2), gi(\theta_1, \theta_2)],$$

and

$$\nu(\theta_1, \theta_2) = gi\left[\varphi(\theta_1, \theta_2)^T, \varphi(\theta_1^T, \theta_2^T)^T\right].$$

In [2] we proved that $\{\theta_1, \theta_2\}_* \sqsubseteq K(\theta_1, \theta_2)_{/\sqsubseteq}$ and $K(\theta_1, \theta_2)_{/\sqsubseteq} \sqsubseteq \{\theta_1, \theta_2\}^*$.

The set $K(\theta_1, \theta_2)$ can be used for computing $\{\theta_1, \theta_2\}^*$ "at once" and this feature is exploited in [2] where we introduced the function FC1, (described in Algorithm 1), which implicitly uses the following unique inference rule.

U: from θ_1, θ_2 deduce any triple $\tau \in \{\theta_1, \theta_2\}*$.

Note the $\{\theta_1, \theta\}_*$ is obtained from $K(\theta_1, \theta_2)$ by applying the function FindMaximal.

Actually, FC1 uses $K(\theta_1, \theta_2)$ instead of $\{\theta_1, \theta_2\}_*$ for sake of efficiency.

FC1 can be enhanced by observing [2] that if θ_1' and θ' belong to $\{\theta_1, \theta_2\}_*$, then $\{\theta_1', \theta'\}_*$ is g-included to $\{\theta_1, \theta_2\}_*$. The validity of this observation follows easily since:

$$\{\theta_1', \theta'\}_* \sqsubseteq \{\theta_1', \theta'\}^* \sqsubseteq \{\theta_1, \theta_2\}^* \sqsubseteq \{\theta_1, \theta_2\}_*.$$

Therefore, it is not necessary to apply the inference rule U to a pair θ_1' and θ', generated by U from the same two triples θ_1 and θ_2, since θ_1' and θ' generate only redundant triples, which would be discarded by the function FindMaximal.

Algorithm 1: Fast closure by U

```
1: function FC1 (J)
2:     J_0 ← J
3:     N_0 ← J
4:     k ← 0
5:     repeat
6:         k ← k + 1
7:         N_k := ⋃_{θ1∈J_{k−1}, θ2∈N_{k−1}} K(θ1, θ2)
8:         J_k ← FindMaximal (J_{k−1} ∪ N_k)
9:     until J_k = J_{k−1}
10:    return J_k
11: end function
```

For the same reasons, we do not need to apply the rule U between a triple θ and another one θ' generated from θ (by combining θ with another triple θ''): in fact if $\theta' \in \{\theta, \theta''\}_*$, then $\{\theta, \theta'\} \subseteq \ ''\}^*$ and so:

$$\{\theta, \theta'\}_* \sqsubseteq \{\theta, \theta''\}_*,$$

which implies that no maximal triple can be obtained.

Then, the use of the inference rule U in FC1 can be enhanced by keeping track of the "parents" of each triple and by neglecting the pairs which satisfy the two previously described situations ("sibling" triples and "father–child").

3. Experimental results

In this section we shortly discuss some experimental results obtained with an implementation in C++ of the algorithm FC1, as well as an implementation of an algorithm to compute the complete closure (with respect to G1–G5). The main purpose of these experiments is to prove the viability of the fast closure computation. Some preliminary results, with different experimental settings, have already been given in [2].

The first aspect, that arises from experiments, is to show the computational cost of the fast closure. In fact, this problem is a computationally hard problem, for which no efficient (i.e. polynomial time) solution can exist as already noted in [34,35]. Therefore an empirical evaluation is necessary in order to establish whether the computation of the fast closure is reasonably fast and uses an acceptable amount of memory.

The other question is to provide a comparison in size and in computation time of the fast closure with respect to the complete closure. The fast closure is clearly smaller than the complete closure (each triple $\theta \in J_*$ corresponds to several triples in $\bar{J}$), but we have not been able to find any theoretical bounds for the size of J_* with respect to the size of $\bar{J}$.

The experiments were performed on an AMD Dual Core Opteron running at 1.8 GHz with 2 GByte main memory. We applied a cut-off of 5,000,000 triples that can be stored (to avoid problems with memory) and a time-out of 3600 s.

In the first set of experiments, we have generated *200* random sets of triples having v variables and r triples, for $r = 10$, 15, 20, 25, 30 and $v = \lfloor 0.5 \cdot r \rfloor$, r, $\lfloor 1.5 \cdot r \rfloor$, $2r$. and we have computed the fast closure by means of FC1.

In the Table 1, the value *perc* is the percentage of the sets for which FC1 has been able to compute the fast closure, within the limits of time and memory, *time* is the average computation times in seconds, *size* is the average size of the fast closure, *iter* is the average number of iterations needed to find the closure, and *gen* is the average number (rounded to the nearest integer) of the overall generated triples.

The behavior of FC1, as explained in the following, is influenced by many factors. Note that as r grows, instances with a small value for $\frac{v}{r}$ become more difficult: with $r = 30$ and $v = 15$ FC1 has not been able to solve any instance. On the other hand, when the ratio $\frac{v}{r}$ is large, instances get easier to solve. In this experiments we have no evidence on how sharp the transition from difficult to easy instances is. It would be investigated in a future work.

The first behavior can be explained as follows: by generating instances with few variables, with respect to the number of relations, can produce many triples where it is possible to repeatedly apply the generalized inference rules. In these cases, the computation of the fast closure requires several iterations and a large number of triples can be generated (most of them are discarded). These kinds of instances seem to be the hardest to solve, in comparison with other kinds.

Table 1
Fast closure FC1.

r	*v*	*perc*	*time*	*size*	*iter*	*gen*
10	5	100	0	10.83	3.99	202
10	10	100	1.06	95.93	6.42	27524
10	15	99	44.43	226.08	6.263	241219
10	20	98.5	22.16	153.54	4.81	115006
15	7	100	9.11E−02	46.84	5.50	5841
15	15	63	500.42	982.68	10.03	1926990
15	22	80.5	111.49	365.29	6.63	359213
15	30	98	9.77	72.14	3.25	32615
20	10	100	79.19	433.835	7.41	652608
20	20	27.5	376.43	921.47	10.2	1105693
20	30	93.5	84.64	305.21	5.58	240052
20	40	98.5	3.64	54.95	2.20	16514
25	12	49.5	1383.23	1354.33	8.3	5231558
25	25	35	254.46	719.69	9.04	720993
25	37	97.5	14.25	124.42	3.8	62761
25	50	100	1.1E−03	29.685	1.445	84
30	15	0	–	–	–	–
30	30	51.28	118.59	514.58	7.65	3631898
30	45	100	0.03	48.38	2.41	1063
30	60	100	8.55E−05	31.06	1.12	7

At the same time, when the number of variables becomes too large, since we generate triples which could contain disjoint set of variables and in this case the opportunity for applying the inference rules becomes very low. This is proved by the average size of the fast closure (which is roughly similar to *nr*) and the number of generated triples (which is rather small). In these cases, the closure often coincides (or almost coincides) with the initial set of triples and therefore can be computed with a little computational effort.

In the second set of experiments we compare the computation time needed for finding the complete closure and its size with respect to the time and size of the fast closure. The complete closure is obtained by using an algorithm similar to FC1, which uses all the inference rules G1–G5, without calling FindMaximal. Furthermore, we did not apply for it any cut-off with respect to the number of triples.

Table 2
Fast closure with FC1.

r	*v*	*time*	*size*	*iter*	*gen*	*res*
4	4	0	3.95	2.75	12.1	20
4	6	0	5.85	2.95	29.2	20
7	7	2E−03	18.65	4.95	559.25	20
7	10	1.8E−02	32.05	4.7	1756.15	20
10	10	0.6755	86.9	5.95	18415	20
10	15	42.7225	320.45	6.7	335910.5	20

Table 3
Complete Closure.

r	*v*	*time*	*size*	*iter*	*gen*	*res*
4	4	0	64	7	57	20
4	6	0.05	527	8.9	899	20
7	7	1.75	3282	13.15	9526	20
7	10	248	28808	13.89	147249	19
10	10	603	50760	16.67	268381	15
10	15	3513	159164	14	683991	1

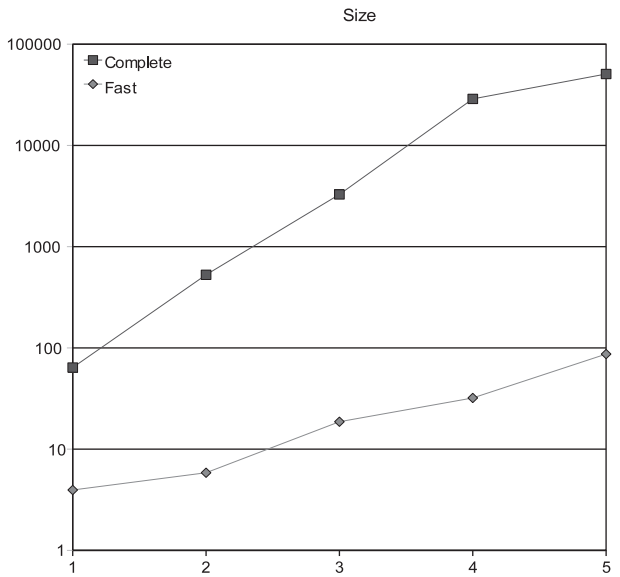

Fig. 1. Comparison of average sizes of the closure.

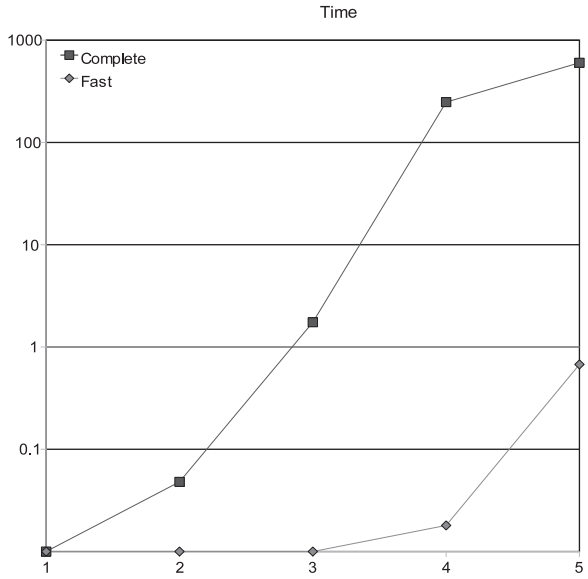

Fig. 2. Comparison of average computation times.

Since we expect that the complete closure is much larger than its fast version, we have performed these new experiments with smaller instances, instead of using the previous one. In particular, we generate 20 sets of r triples and v variables, for $r = 4, 7, 10$ and $v = r, \lfloor 1.5 \cdot r \rfloor$. Note that all the instances are solved by FC1 while this does not happen for complete closure.

In Table 2 the results for the fast closure are reported, the average values by FC1: the average computation time is negligible, except that in the last row, where we obtain results similar in magnitude order, as those displayed in Table 1. The algorithm FC1 has been able to build the closure for each instance.

In Table 3 we show the results related to the complete closure. The last column contains the number of instances for which the algorithm has been able to compute the complete closure within an hour of computation. Note that for $r = 10$ and $v = 15$ we could solve only one instance, which almost reached the time limit, while the fast closure of this instance has only 27 triples and has been found in a negligible amount of time. The values in the last column are used to compute the average values shown in Table 2.

The comparison of the size between fast and complete closure is impressive, as shown in Fig. 1 (the last rows of both tables have been ignored).

Clearly also the computation times for computing the complete closure are much higher than the time needed to compute the fast closure (see Fig. 2).

4. Graphs and maps

In this section we recall some notions about graphs and the representation of an independence model by an acyclic directed graph (DAG) [26]. We denote by $G = (\mathcal{U}, \mathcal{E})$ a graph with a set $\mathcal{U}$ of nodes and a set $\mathcal{E}$ of directed arcs. For any $u \in \mathcal{U}$, as usual, we denote with $pa(u)$ the parents of u, $ch(u)$ the children of u, $ds(u)$ the sets of descendants and $an(u)$ the set of ancestors. We use the convention that each node u belongs to $an(u)$ and to $ds(u)$, but not to $pa(u)$ and $ch(u)$.

Definition 2. If A, B and C are three disjoint subsets of nodes in a DAG G, then C is said to d-separate A from B (in symbol $(A,B,C)_G$) if for each non-directed path between a node in A and a node in B, there exists a node x in the path which satisfies one of these two conditions:

(1) x is a collider (i.e. both edges point to x), $x \notin C$ and $ds(x) \cap C = \emptyset$;
(2) x is not a collider and $x \in C$.

In order to study the representation of a conditional independence model, we need to distinguish between dependence maps and independence maps, since there are conditional independence models that cannot be completely represented by a DAG (see e.g. [21,26,35]).

Definition 3. Let J be a set of conditional independence relations on S. A DAG $G = (S, \mathcal{E})$ is a dependence map (briefly a D-map) for J if for each triple $(A,B,C) \in S^{(3)}$:

$$(A,B,C) \in \bar{J} \Rightarrow (A,B,C)_G.$$

Moreover, $G = (S, \mathcal{E})$ is an independence map (briefly an I-map) for $\bar{J}$ if for each triple $(A,B,C) \in S^{(3)}$:

$$(A,B,C)_G \Rightarrow (A,B,C) \in \bar{J}.$$

G is a minimal I-map of $\bar{J}$ if deleting any arc, G is no more an I-map.G is said to be a perfect map (briefly a p-map) for $\bar{J}$ if it is both a I-map and a D-map.

The next definition and theorem [26] provide a method to build a DAG given an independence model $\bar{J}$.

Definition 4. Let $\bar{J}$ be an independence model defined on S and let $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ be an ordering of the elements of S. The boundary strata of $\bar{J}$, relative to π, is an ordered set of subsets $\langle B^\pi_{(1)}, B^\pi_{(2)}, \ldots, B^\pi_{(n)} \rangle$ of S such that each $B^\pi_{(i)}$ is a minimal set satisfying $B^\pi_{(i)} \subseteq S^\pi_{(i)} = \{\pi_1, \ldots, \pi_{i-1}\}$ and either $B^\pi_{(i)} = S^\pi_{(i)}$ or $\gamma_i = (\{\pi_i\}, S^\pi_{(i)} \setminus B^\pi_{(i)}, B^\pi_{(i)}) \in \bar{J}$.

The DAG created by setting each $B^\pi_{(i)}$ as parent set of the node π_i is called boundary DAG of J, relative to π.

It is important to notice that, under graphoid axioms, the minimal set $B^\pi_{(i)}$ is unique (see [26]).

Moreover, Definition 4 is a reformulation of that given in [26]: we have (as in the cited paper) a minimal set for each nodes, but some of them would generate an illegal triple (i.e. a triple which does not belong to $S^{(3)}$) and then they could not be in $\bar{J}$.

In the rest of the paper, $S^\pi_{(i)}$ and $B^\pi_{(i)}$ will be denoted, when it is clear from the context, simply as $S_{(i)}$ and $B_{(i)}$, respectively. The triple γ_i is known as *basic triple*.

The next theorem is an extension of Verma's Theorem [41] stated for conditional independence relations (see [26]).

Theorem 1. *Let J be an independence model closed with respect to the graphoid properties. If G is a boundary DAG of J, relative to any ordering π, then G is a minimal I-map of J.*

Theorem 1 helps to build a DAG for an independence model $\bar{J}$ (induced by a probability P on S) given an ordering π on S. In the rest of the paper, given an ordering π on S, G_π is the corresponding I-map of $\bar{J}$ with respect to π.

Finally it is worth to notice that Theorem 1 can also be formulated for semi-graphoid, but this is out of the aims of this paper.

5. BN-DRAW function

The aim of this section is to review the procedure BN-DRAW introduced in [3], which builds the minimal I-map G_π of $\bar{J}$ (see Definition 3) given the fast closure J_* (introduced in Section 2.1) of J and an ordering π on S.

Note that the standard procedure to draw an I-map (see [19,26]), described in Definition 4, cannot be applied to J_*. In fact, as shown in Example 1, the basic triples, related to an arbitrary ordering π, could not belong to J_*, but they could be just g-included into some triples of J_*.

Example 1. Given $J = \{(\{1\},\{2\},\{3,4\}), (\{1\},\{3\},\{4\})\}$, the aim is to find the corresponding basic triples and to draw the relevant DAG G_π related to the ordering $\pi = \langle 4,2,1,3 \rangle$ (see Fig. 3). By the closure with respect to graphoid properties we obtain:

$$\begin{aligned}\bar{J} = \{&(\{1\},\{2\},\{3,4\}), (\{1\},\{3\},\{4\}), (\{1\},\{2,3\},\{4\}), (\{1\},\{2\},\{4\}), (\{1\},\{3\},\{2,4\}),\\ &(\{2\},\{1\},\{3,4\}), (\{3\},\{1\},\{4\}), (\{2,3\},\{1\},\{4\}), (\{2\},\{1\},\{4\}), (\{3\},\{1\},\{2,4\})\}\end{aligned}$$

and the set of basic triples (related to π) is

$$\Gamma = \{(\{1\},\{2\},\{4\}), (\{3\},\{1\},\{2,4\})\}.$$

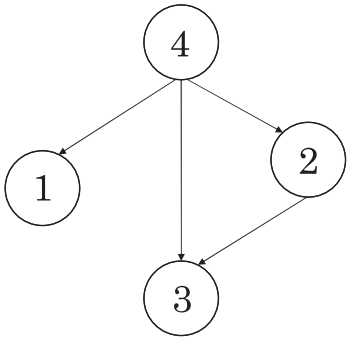

Fig. 3. I-map related to Examples 1 and 2.

By FC1 we obtain $J_* = \{(\{1\},\{2,3\},\{4\})\}$ and it is simple to observe that $\Gamma \sqsubseteq J_*$.

However, we show in Proposition 2 how to extract the basic triples from the fast closure.

Proposition 2. *Let J be a set of independence relations on S, J_* its fast closure and π an ordering on S. For each $i \in S$, the set:*

$$\mathcal{B}_i = \left\{(\{i\},B,C) \in S^{(3)} : B \cup C = S_{(i)}, \exists \theta \in J_* \quad \text{with} \quad (\{i\},B,C) \sqsubseteq \theta\right\}$$

is not empty if and only if the basic triple $\gamma_i = (\{i\}, S_{(i)} \setminus B_{(i)}, B_{(i)})$ exists (i.e. $\gamma_i \in S^{(3)}$), and coincides with the unique maximal triple $\bar{\gamma}_i$ of $\mathcal{B}_i$.

Proof. Suppose that $\mathcal{B}_i$ is not empty. Let us show that in $\mathcal{B}_i$ there is a unique maximal triple. If $\mathcal{B}_i$ has only one element, the claim is trivially true. Otherwise, for each $\theta', \theta'' \in \mathcal{B}_i$, it is possible to apply the rule G5* (Generalized intersection) to θ', θ'', and we obtain θ'''. Now, θ''' belongs to $\mathcal{B}_i$: in fact, θ''' is g-included in a triple obtained by applying G5* to some pairs of triples in J_*. Moreover, both θ' and θ'' are g-included in θ'''. Note that $|C'''| < |C'|$ and $|C'''| < |C''|$. By iterating this process, we terminate (since S is finite) by finding a maximal triple $\bar{\gamma}_i$ of $\mathcal{B}_i$. This triple is clearly unique and coincides with γ_i. In fact, by supposing the contrary by definition of γ_i, it would follow that $|B_i| < |B_{(i)}|$. But this is impossible, otherwise there should exist a triple in J_* g-including $\bar{\gamma}_i$.

Vice versa, if the basic triple $\gamma_i = (\{\pi_i\}, S_{(i)} \setminus B_{(i)}, B_{(i)})$ for π_i exists, then it is straightforward to see that $\gamma_i \in \mathcal{B}_i$. □

In order to describe a new version of BN-DRAW we need to introduce the following operation: for each $\theta = (A,B,C) \in S^{(3)}$, for any $x \in S$ and $T \subseteq S$, define:

$$\Pi(\theta,T,x) = \begin{cases} T \cap (A \cup C) & \text{if } C \subseteq T \subseteq A \cup B \cup C \quad \text{and} \quad x \in A \\ T \cap (B \cup C) & \text{if } C \subseteq T \subseteq A \cup B \cup C \quad \text{and} \quad x \in B \\ T & \text{otherwise.} \end{cases}$$

Then, the function $\Pi(\theta, S_{(x)}, x)$ computes the set of potential parents of x in G_π when the basic triple γ_x is g-included in θ. In the case that no basic triple is g-included in θ, then $\Pi(\theta, S_{(x)}, x) = S_{(x)}$.

The function PARENTS, as shown in Algorithm 2, finds the smallest set of potential parents.

Algorithm 2: The set of parents of x

```
function PARENTS (x, T, K)
  pa ← T
  for all θ ∈ K do
    p ← Π(θ, T, x)
    if |p| < |pa| then pa ← p
  end for
  return pa
end function
```

Note that the procedure BN-DRAW applies, for each π_i, the function PARENTS in order to find the parent set of π_i.

Given π, BN-DRAW builds the minimal I-map G_π in linear time, with respect to the cardinality m of J_* and the number n of random variables. In fact, it is based on the function PARENTS, which computes the set of parents of a given variable in $O(m)$ steps. In each step, some set operations must be executed and this can be efficiently performed by using a compact representations for sets (e.g., as bit vectors). The space needed in memory by BN-DRAW is almost exclusively used to store the fast closure.

Algorithm 3: DAG from J_* given an ordering π of S

```
function BN-DRAW (n, π, J_*)
  T ← ∅
  G ← a graph with S as vertex set and no edges
  for i ← 2 to n do
      T ← T ∪ {π_{i−1}}
      pa ← PARENTS (π_i, T, J_*) draw an arc in G from each index in pa to π_i
  end for
  return G
end function
```

The next example compares the standard procedure recalled in Definition 4 with BN-DRAW to build the *I*-map, given a subset J of $S^{(3)}$ and an ordering π among the elements of S.

Example 2. Consider the same independence set J of Example 1 and the ordering $\pi = \langle 4,2,1,3\rangle$, we compute the basic triple by applying BN-DRAW to $J_* = \{\theta\}$ with $\theta = (\{1\},\{2,3\},\{4\})$.

For $i = 2$, $S_{(2)} = \{4\}$, $2 \in B$ and $\Pi(2,\{4\},\theta) = \{4\}$. For $i = 1$, $S_{(1)} = \{2,4\}$, $1 \in A$ and $\Pi(1,\{2,4\},\theta) = \{4\}$. For $i = 3$, $S_{(3)} = \{1,2,4\}$, $3 \in B$ and $\Pi(3,\{1,2,4\},\theta) = \{2,4\}$.

The basic triple for 2 does not exist because $B \cap S_{(2)} = \emptyset$, while for 1 and 3 the corresponding basic triples are respectively $(\{1\},\{2\},\{4\})$ and $(\{3\},\{1\},\{2,4\})$. Therefore, we obtain the same set Γ given in Example 1 and the *I*-map shown in Fig. 3.

6. Existence of Perfect map

6.1. Sufficient condition

We recall, for sake of completeness, the result described in [3], which ensures the existence of a perfect map, by using the notation of this paper. For the proof see the quoted paper, however it can also be easily deduced from the proof of the sufficient implication of Theorem 2. Nevertheless, this result is interesting because the latter is a generalization of the former.

Proposition 3. *Let J be a set of conditional independence relations. Given an ordering π on S, let G_π be the corresponding I-map. If for any triple $\theta = (A,B,C) \in J_*$ the following conditions hold (where $\mathcal{X} = A \cup B \cup C$):*

(1) $S_{(c)} \cap A = \emptyset$ *or* $S_{(c)} \cap B = \emptyset$ *for each* $c \in C$;
(2) $S_{(x)} \subseteq \mathcal{X}$ *for each* $x \in \mathcal{X}$;

then the related I-map G_π is a perfect map.

Next examples show that even if the conditions (1) or (2) of Proposition 3 are not satisfied, a perfect map could exist.

Example 3. Let us consider the set:

$$J = \{(\{2\},\{5\},\emptyset),(\{4\},\{5\},\{2\}),(\{1\},\{2\},\{4,5\}),(\{3\},\{4,5\},\{1,2\})\}.$$

The associated fast closure is $J_* = \{\theta_1,\ldots,\theta_4\}$ with:

$$\theta_1 = (\{3\},\{4,5\},\{1,2\}),$$
$$\theta_2 = (\{5\},\{2,4\},\emptyset),$$
$$\theta_3 = (\{2\},\{1,5\},\{4\}),$$
$$\theta_4 = (\{5\},\{2,3\},\{1,4\}).$$

The *I*-map G_π (see Fig. 4), related to the ordering $\pi = \langle 2,4,5,1,3\rangle$, is a perfect map. Anyway, Proposition 3 does not hold. Indeed, by looking for orders satisfying condition (2), the only possibilities consist into taking any permutation of $\{2,4,5\}$

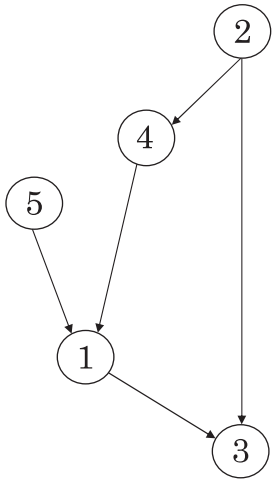

Fig. 4. *P*-map related to Example 3.

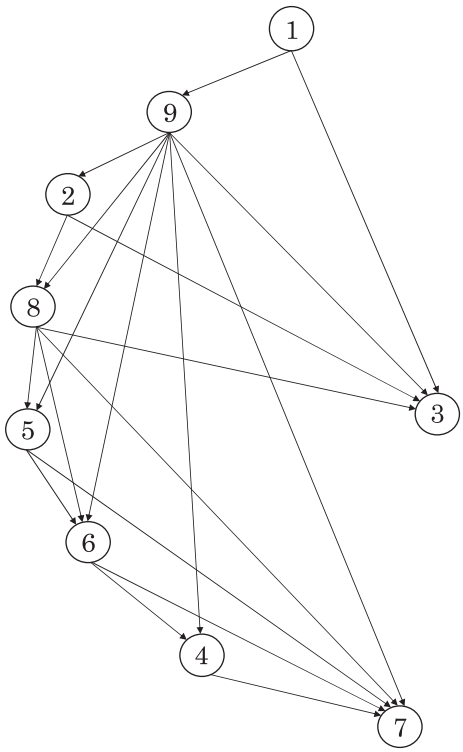

Fig. 5. *P*-map related to Example 4.

followed by 1 and then by 3. In all these 6 orderings, condition (1) is always violated for θ_4, because $S_{(1)} = \{2,4,5\}$, $S_{(1)} \cap \{5\} \neq \emptyset$, and $S_{(1)} \cap \{2,3\} \neq \emptyset$.

The following example shows that Condition (*2*) of Proposition 3 is too strong.

Example 4. Let us consider the set $J = \{\theta_1, \theta_2\}$ with $\theta_1 = (\{1,2,3\},\{4,5,6,7\},\{8,9\})$ and $\theta_2 = (\{1,4\},\{2,5,8\},\{6,9\})$. The related fast closure is $J_* = \{\theta_1, \theta_2, \ldots, \theta_7\}$ with:

$$\theta_3 = (\{1\},\{2,4,5,6,7,8\},\{9\}),$$
$$\theta_4 = (\{2\},\{1,4,5,6,7\},\{8,9\}),$$
$$\theta_5 = (\{4\},\{1,2,3,5,8\},\{6,9\}),$$
$$\theta_6 = (\{5\},\{1,2,3,4\},\{6,8,9\}),$$
$$\theta_7 = (\{2,4\},\{1,5\},\{6,8,9\}).$$

Conditions (2) of Proposition 3 do not hold: in fact, by considering the triples θ_3 and θ_5 it is simple to observe that $7 \in \mathcal{X}_3$, but $7 \notin \mathcal{X}_5$ and $3 \notin \mathcal{X}_3$, while $3 \in \mathcal{X}_5$. Then, there is no ordering π satisfying conditions *(1)* and *(2)* of Proposition 3 for any $\theta \in J_*$.

Nevertheless, by considering the ordering $\pi = \langle 1,9,2,8,3,5,6,4,7\rangle$, the related *I*-map G_π (represented in Fig. 5) is perfect, i.e. it represents each triple of J_*.

6.2. A necessary and sufficient condition

Now, we provide one of the main results of this paper, which is a necessary and sufficient condition for the existence of a perfect map. This result gives a characterization of those orderings generating a perfect map. This result has been obtained by studying the violations of conditions in Proposition 3 in the cases where a perfect map does exist. Indeed, each time the conditions (1) or (2) are not satisfied, the existence of a perfect map is guaranteed if a weaker condition is met.

Theorem 2. *Let J be a set of independence relations. Then, $\bar{J}$ is representable by a p-map if and only if there exists an ordering π such that for each $\theta = (A,B,C) \in J_*$ the following conditions hold:*

C1: *for each $c \in C$ such that $S_{(c)} \cap A \neq \emptyset$ and $S_{(c)} \cap B \neq \emptyset$, there exists a triple $\theta_c \in J_*$ such that $\Pi(\theta_c, S_{(c)}, c) \cap A = \emptyset$ or $\Pi(\theta_c, S_{(c)}, c) \cap B = \emptyset$;*

C2: *for each* $a \in A$ *such that* $S_{(a)} \cap B \neq \emptyset$ *or* $S_{(a)} \cap (S \setminus \mathcal{X}) \neq \emptyset$ *there exists a triple* $\theta_a \in J_*$ *such that* $\Pi(\theta_a, S_{(a)}, a) \cap [B \cup (S \setminus \mathcal{X})] = \emptyset$;
C3: *for each* $b \in B$ *such that* $S_{(b)} \cap A \neq \emptyset$ *or* $S_{(b)} \cap (S \setminus \mathcal{X}) \neq \emptyset$ *there exists a triple* $\theta_b \in J_*$ *such that* $\Pi(\theta_b, S_{(b)}, b) \cap [A \cup (S \setminus \mathcal{X})] = \emptyset$;
C4: *for each* $c \in C$ *such that* $S_{(c)} \cap (S \setminus \mathcal{X}) \neq \emptyset$, *there exists a triple* $\theta'_c \in J_*$ *such that* $\Pi(\theta'_c, S_{(c)}, c) \cap (S \setminus \mathcal{X}) = \emptyset$.

where $\mathcal{X} = (A \cup B \cup C)$.

Proof. ($\Rightarrow$) Let us suppose that G_π is a p-map for J_*. We need to prove that π satisfies the conditions $C1$–$C4$.

Let $\theta = (A,B,C)$ be a triple in J_*. Under the hypotheses, θ is represented in G_π. Let us prove by absurd that all the conditions $C1$–$C4$ are satisfied for θ.

If condition *C1* is not satisfied, then there exists an index $c \in C$, with $S_{(c)} \cap A \neq \emptyset$ and $S_{(c)} \cap B \neq \emptyset$, but, for all triples $\theta' \in J_*$, it happens that $\Pi(\theta', S_{(c)}, c) \cap A \neq \emptyset$ and $\Pi(\theta', S_{(c)}, c) \cap B \neq \emptyset$. Hence, there exists an element α in $pa(c) \cap A$ and an element β in $pa(c) \cap B$ such that the path $\alpha \to c \leftarrow \beta$ is not blocked by C, contradicting the fact that A is d-separated from B by C.

If condition *C2* is not satisfied, then there exists an index $a \in A$, with $S_{(a)} \cap B \neq \emptyset$ or $S_{(a)} \cap (S \setminus \mathcal{X}) \neq \emptyset$, but, for all triples $\theta' \in J_*$, it happens that $\Pi(\theta', S_{(a)}, a) \cap [B \cup (S \setminus \mathcal{X})] \neq \emptyset$. Hence, there exists either an element β in $pa(a) \cap B$ or an element δ in $pa(a) \cap (S \setminus \mathcal{X})$. In the former case, there exists an arc between an element of A and an element of B, and θ is not represented in the graph. In the latter case, we prove that $\bar{\theta} = (\{\delta\} \cup A, B, C)$ is represented in G_π. Any path from δ to an element of B is blocked by C, otherwise that path could be extended to a, but then it would not be blocked by C (δ is not collider). But this is impossible because, being G_π a p-map, $\bar{\theta} \in \bar{J}$ and $\theta \sqsubseteq \bar{\theta}$, so this contradicts the maximality of θ.

The proof of the validity of condition *C3* goes along the same line.

If condition *C4* is not satisfied, then there exists an index $c \in C$, with $S_{(c)} \cap (S \setminus \mathcal{X}) \neq \emptyset$, but, for all triples $\theta' \in J_*$, it happens that $\Pi(\theta'_c, S_{(c)}, c) \cap (S \setminus \mathcal{X}) \neq \emptyset$. Hence, there exists an element ϵ in $pa(c) \cap (S \setminus \mathcal{X})$. Now, either $\theta' = (\{\epsilon\} \cup A, B, C)$ or $\theta'' = (A, \{\epsilon\} \cup B, C)$ is represented in G_π, otherwise, there would be a path ρ' from $b \in B$ to ϵ and a path ρ'' from ϵ to $a \in A$ not blocked by C. Indeed, the path obtained by concatenating ρ' to ρ'' connects B to A and it is not blocked by C, since ϵ is a parent of C. Being G_π a p-map for $\bar{J}$, therefore also for J_*, we have that $\theta' \in \bar{J}$ or $\theta'' \in \bar{J}$. Since, by construction, $\theta \sqsubseteq \theta'$ and $\theta \sqsubseteq \theta''$, this contradicts the maximality of θ in $\bar{J}$.

($\Leftarrow$) It is easy to see that conditions $C1$–$C4$ imply that for each $\theta = (A,B,C) \in J_*$ in G_π the following conditions hold:

(a) for each $a \in A$, $pa(a) \subseteq A \cup C$;
(b) for each $b \in B$, $pa(b) \subseteq B \cup C$;
(c) for each $c \in C$, either $pa(c) \subseteq A \cup C$ or $pa(c) \subseteq B \cup C$.

As a consequence, for each $x \in \mathcal{X}$, $pa(x) \subseteq \mathcal{X}$.

Now, let us show that each $\theta = (A,B,C) \in J_*$ is represented in G_π. Let $\rho = (u_1, \ldots, u_l)$ be a path in G_π from $u_1 \in A$ to $u_l \in B$. Put $j = \max\{i\colon u_i \in A\}$ and $l = \min\{i\colon u_i \in B\}$. Then, $j + 1 \leqslant l - 1$, otherwise there would be an element of A having parents in B or vice versa. If $u_{j+1} \in pa(u_j)$, then $u_{j+1} \in C$ and, since it is not a collider, it blocks ρ. Similarly, if $u_{l-1} \in pa(u_l)$.

Now, let us suppose that $u_{j+1} \in ch(u_j)$ and $u_{l-1} \in ch(u_l)$. Let r be an index such that u_i precedes u_r in π for each $i = j, \ldots, l$. Clearly $r \neq j$ and $r \neq l$. Moreover, u_r is a collider. If $u_r \in C$, then it is impossible that $j + 1 = r = l - 1$, otherwise u_r would have parents both in A and in B. So, at least one between u_{r-1} and u_{r+1} is a parent of u_r and then blocks ρ, since belongs to C.

On the other hand, if $u_r \notin C$, then it is impossible that a descendent of u_r belongs to C (because all ancestors of elements of C belong to $\mathcal{X}$). Therefore u_r blocks ρ. □

7. Implementation and experimental result

In this section we describe an algorithm which is based on Theorem 2 to check whether a set J_* is representable by a graph, and in the affirmative case, to find a perfect map.

This algorithm is not based on a direct implementation of Theorem 2, because it would require the validity of conditions $C1$–$C4$ for every possible ordering among the variables.

Hence, a not naive implementation exploits two important features of conditions $C1$–$C4$. First of all, they can be used for building a partial ordering. If the ordering is built in a sequential way, i.e. selecting the first variable, then the second one, and so on, it is possible to determine, at each step, the set $S_{(x)}$ for the last chosen variable x. Then, the conditions $C1$–$C4$ can be checked only for x, by verifying, only for each triple $\theta \in J_*$ where x appears, the existence of a suitable triple θ_x (in the cases requested by each condition).

However, it is not necessary to search such triples. Indeed, from $S_{(x)}$ it is possible to compute the parent set $pa(x)$ of x by means of function PARENTS. Hence, for instance, to test Condition *C1* for x it is sufficient to check for each triple $\theta = (A,B,C)$, such that $x \in C$, whether $pa(x) \cap A = \emptyset$ or $pa(x) \cap B = \emptyset$. In fact, if $pa(x) \cap A = \emptyset$ or $pa(x) \cap B = \emptyset$ are true, then the triple θ' from which $pa(x)$ is taken (i.e. $pa(x) = \Pi(\theta', S_x, x)$), is such that $\Pi(\theta', S_x, x) \cap A = \emptyset$ or $\Pi(\theta', S_x, x) \cap B = \emptyset$. On the other hand, if for some triple θ' it happens that $\Pi(\theta', S_x, x) \cap A = \emptyset$ or $\Pi(\theta', S_x, x) \cap B = \emptyset$, then the same relation holds for $pa(x)$ since it is a subset of $\Pi(\theta', S_x, x)$. The same consideration holds for *C2, C3* and *C4*.

These remarks greatly simplify the algorithm, because the computation time needed to check all the conditions reduces to $O(m)$ instead of $O(m^2)$ as required by a direct implementation, where m is the cardinality of J_*.

In the affirmative case, the procedure continues with the remaining variables, extending the ordering. Note that the conditions need not be verified again for x. In the negative case, any total ordering which extends the current partial order will not satisfy the conditions. Therefore the procedure comes back to the previous position and makes a new choice for x. In this way, conditions make an early pruning on the orderings and thus it is not necessary to search in the space of total orderings.

The other important feature is the particular structure of *C2, C3* and *C4*. Let us examine as an example, condition *C2*. For a given triple $\theta = (A,B,C)$, no element of $B \cup (S \setminus \mathcal{X})$ can be parent of any $a \in A$, otherwise *C2* would be violated. The conditions *C3* and *C4* behave in a similar way. This fact can be exploited by computing, before starting the search process, the set of *forbidden parents*, denoted as $NP(x)$, for each $x \in S$, as shown in Algorithm 4. Note that the particular form of condition *C1* (it is a disjunction and A and B are disjoint) makes impossible to exclude any variable from being a parent of other variables. Hence the preprocessing phase (Algorithm 4) does not take into account condition *C1*.

Algorithm 4: Preprocessing for conditions *C*2–*C*4

```
function PREPROCESS (K)
    for all x ∈ S : NP(x) ← ∅
    for all θ = (A,B,C) ∈ K do
        X ← A ∪ B ∪ C
        R ← S \ X
        for all x ∈ A:NP(x) ← NP(x) ∪ B ∪ R
        for all x ∈ B:NP(x) ← NP(x) ∪ A ∪ R
        for all x ∈ C:NP(x) ← NP(x) ∪ R
    end for
end function
```

Once the element x is selected and $pa(x)$ is computed, conditions *C2–C4* are checked by simply verifying that $pa(x)$ does not contain any element of $NP(x)$. Hence, the filter for the partial order is described in Algorithm 5.

Algorithm 5: Checking conditions *C1–C4*

```
function CHECK–CONDS (π,i,K)
    T ← π[1,...,i − 1]
    x ← π_i
    Q ← PARENTS (x,T,K)
    if NP(x) ∩ Q ≠ ∅ then return FALSE
    for all θ = (A,B,C) ∈ K do
        if (x ∈ C) ∧ (Q ∩ A ≠ ∅) ∧ (Q ∩ B ≠ ∅) then return FALSE
    end for
    return TRUE
end Function
```

The rest of the algorithm, which we propose here, is a standard backtracking-based search procedure, as described in Algorithm 6, in which the symbol $\emptyset$ denotes an empty sequence of integers.

Algorithm 6: Main function for representability

```
function REPRESENT (J_*)
    PREPROCESS (J_*)
    return SEARCH (∅,1,S,J_*)
end function
```

The recursive function SEARCH incrementally tries to build an ordering π satisfying conditions *C1–C4* of Theorem 2. It returns the element $\perp$ if it fails into finding such an ordering. At the ith recursive call it attempts to choose the ith element in π, by selecting each of the remaining variables. A simple strategy, which somehow implements the well known principle *fail first*, is to choose the not already selected variable x having the largest value of $|pa(x)|$. For each possible variable x, the procedure CHECK–CONDS checks whether the conditions *C1–C4* are not violated by setting π_i as x. In the positive case, it calls itself until a complete ordering is obtained. If no variable can be set at the ith place of π, then the recursive call fails and a revision of the previously chosen variables is performed (backtracking).

The number of the steps of SEARCH is in the worst case exponential in n, but as already seen backtracking can perform an early pruning on orderings. Finally, note that SEARCH can avoid at all to call BN-DRAW by storing, for each $x \in S$, the sets Q computed in the function CHECK–CONDS.

The algorithm has been implemented in C++ and has been tested, to show the potential applicative use of Theorem 2.

The experimental setting is the following: for each $n = 10,15,\ldots,40$ and $p_e = 0.2, 0.4, 0.6, 0.8$ we have generated $N_g = 25$ random DAGs having n nodes and p_e as edge occurring probability. Then, for each DAG G, REPRESENT has been called on Γ_*, where Γ is the set of the basic triples extracted from G. The expected answer is hence always positive and the obtained graph must be equivalent [29] to G. The computer used is the same as the experiments described in Section 3. However, the time-out was now set to 10,800 s.

Algorithm 7: Backtracking procedure

```
function SEARCH (π, i, V, K)
    if V = ∅ then
        return BN-DRAW (π, K)
    else
        for all x ∈ V do
            π_i ← x
            if CHECK–CONDS (π, i, K) then
                G ← SEARCH(π, i + 1, V\{x}, K)
                if G ≠ ⊥ then return G
            end if
        end for
        return ⊥
    end if
end function
```

In Table 4 we report for each combination of n and p_e: the average number of the basic triples $|\Gamma|$, the average size of the fast closure $|\Gamma_*|$, the average time $T1$ to compute Γ_*, the number of instances solved within the time-out nr and finally the average computation time $T2$ spent by REPRESENT.

Table 4
Experimental result for graph searching.

n	p_e	$\lvert\Gamma\rvert$	$\lvert\Gamma_*\rvert$	$T1$	nr	$T2$
10	0.2	8.72	67.28	0.5	25	0.03
10	0.4	8.36	29.32	0.01	25	0
10	0.6	7.56	11.76	0	25	0
10	0.8	5.48	6.0	0	25	0
15	0.2	13.76	350.76	22.1	25	17.05
15	0.4	13.44	88.56	0.17	25	0.02
15	0.6	12.36	30.68	0	25	0.01
15	0.8	9.6	10.2	0	25	0.01
20	0.2	18.84	1718	1552.53	20	31.17
20	0.4	18.48	249.36	3.7	25	0.1
20	0.6	17.48	57.16	0.02	25	0.03
20	0.8	15.08	17.12	0	25	0.07
25	0.2	23.64	1879	877.98	1	2.26
25	0.4	23.4	592.24	37.84	25	0.75
25	0.6	22.44	83.12	0.05	25	0.11
25	0.8	20.04	24.52	0	25	0.4
30	0.2	28.72	–	–	0	–
30	0.4	28.44	888.12	78.47	25	0.93
30	0.6	27.36	137.84	0.18	25	0.42
30	0.8	24.96	32.2	0	25	2.05
35	0.2	33.84	–	–	0	–
35	0.4	33.44	1601.28	459.44	25	13.45
35	0.6	32.68	173.92	0.23	25	1.32
35	0.8	29.92	41.0	0	25	8.83
40	0.2	38.76	–	–	0	–
40	0.4	38.44	1787.08	462.72	24	51.69
40	0.6	37.8	231.48	0.5	25	4.74
40	0.8	34.6	48.48	0	25	48.85

It is important to notice that for $p_e = 0.2$ the procedure has been able to solve 20 instances on 25 only for $n = 20$, while for higher value of n no instance was solved, except one instance for $n = 25$. This is only due to the time needed to compute the fast closure: the time limit was always reached while computing Γ_*. The reason is that for low values of p_e the basic triples have a large fast closure, while as p_e increases the size of Γ_* strongly decreases. The overall result is that search procedure REPRESENT is faster with respect to fast closure computation, taking only a small part of the total computation time, for sparse graphs, i.e. for $p_e \leqslant 0.4$. On the other hand, as the probability of edge occurrence increases, $T2$ becomes larger than $T1$, which tends to 0.

Another aspect which would be worth to be investigated is that the computation time $T2$ appears to be non monotone with respect to p_e: the values for $p_e = 0.4$ and $p_e = 0.8$ are higher than the value for $p_e = 0.6$. A sort of phase transition phenomenon might be the explanation of this behavior. Anyway, this could be the subject of a future work.

8. Related works and conclusions

In this section we firstly provide a short survey about the problem of representing a set of conditional independencies with a graph. In [28] the problem has been solved for undirected graphs by providing a necessary and sufficient condition which ensures the existence of a graph representing exactly a given model. This condition requires the properties of Symmetry, Decomposition, Intersection and.

- if $(A,B,C) \in J$ then $(A,B,C \cup C') \in J$ for any C' disjoint from A, B and C (Strong Union).
- if $(A,B,C) \in J$ then $(A,\{x\},C) \in J$ or $(\{x\},B,C) \in J$ (Transitivity).

However the representation through an undirected graph is not satisfactory because it is not able to represent "induced" conditional dependencies, i.e. those models for which $(A,B,C) \in J$ but $(A,B,C \cup C') \notin J$.

The representation through a directed acyclic graph does not suffer of this specific problem, even if there are undirect graphs which can faithfully represent conditional dependence models, that cannot be represented by DAGs. In [26] a necessary condition, based on the following statements, for the representability of a model by a DAG is described: Symmetry, Weak Union, Contraction, Intersection and.

(1) $(A,B_1 \cup B_2,C) \in J$ if and only if $(A,B_1,C) \in J$ and $(A,B_2,C) \in J$ (Composition/Decomposition);
(2) if $(A,B,C) \in J$ and $(A,B,C \cup \{c\}) \in J$, then either $(\{c\},B,C) \in J$ or $(A,\{c\},C) \in J$ (Weak Transitivity);
(3) if $(\{a\},\{b\},\{c,d\}) \in J$ and $(\{c\},\{d\},\{a,b\}) \in J$ then either $(\{a\},\{b\},\{c\}) \in J$ or $(\{a\},\{b\},\{d\}) \in J$ (Chordality).

Note that the above condition (1) is a reinforcement of Decomposition rule G2.

These conditions are only necessary, as shown in [26]. Moreover, as already noticed by Pearl [26] and in Geiger's thesis [18] and confirmed by Studený's paper [33], it is unlikely that the DAG representability may have a finite axiomatization.

From the papers of Pearl and Verma [43,30], an algorithm able to verify whether a model admits a p-map has been designed, furtherly refined in [16,32,27]. This algorithm draws an undirected graph (the so called *skeleton*), provides direction to some edges creating the so called *v-structures*, and finally selects the directions of the remaining edges according to some suitable propagation rules, which avoid the formation of loops or other v-structures. The obtained graph is then checked whether it is a p-map. In the negative case, it can be proved that no other graph can represent completely the model.

Another compact way of representing conditional independencies is through annotated graphs, introduced by Paz et al. in 2000 [25] and further extended in [24], where it is provided an algorithm which checks whether the independence relations encoded in an annotated graph can be represented through a DAG.

The problem has been faced in [15] by exploiting the presence of stable independencies [23] in the model, obtaining new necessary conditions, which include all the semi-graphoid properties.

In our work, we provide a mathematical full characterization of sets of independence relations (closed with respect to the graphoid properties) which can be represented through a DAG. As expected, these conditions cannot be reduced in an axiomatic form. Indeed, Theorem 2 requires the existence of a suitable ordering among the variables, which satisfies some conditions, strictly related to the fast closure.

As a corollary of Theorem 2 we also obtain a new characterization of perfect maps (explicitly described in the proof of the theorem), which can be used, as an alternative way with respect to d-separation, to check whether an I-map is also a D-map. Again, this property is expressed in terms of the fast closure.

The algorithm described in Section 7 can be improved in many ways. First of all, by using suitable data structures, we can reduce the time needed to search variables occurring in set of triples, for instance representing J_* as a bipartite graph, where each variable is linked to the triples in which appears and each triple is linked to the variables which contains.

Moreover, the applications of some techniques, used in other constraint satisfaction problems, to the search algorithm can be tested. Indeed, it would be possible to use more enhanced forms of backtracking and variable selection. Also the preprocessing phase could be extended in order to exploit further information from J_*.

Finally, another point of further investigation is the integration of our approach, based on the fast closure, with the theoretical aspects of [30,32]. Indeed, one of the most important drawbacks of Pearl and Verma's algorithm is the need of performing several "queries" on a given model, i.e. to check whether some triple (A,B,C) is implied by J. Even if some of the required queries have a particular form and are computational easy, the overall cost of the algorithm is huge. In our framework, we can exploit the fast closure as a compact representation of the closure and these queries could be quickly solved.

References

[1] B.N. Amor, S. Benferhat, Graphoid properties of qualitative possibilistic independence relations, International Journal of Uncertainty, Fuzziness Knowledge-Based Systems 13 (1) (2005) 59–96.
[2] M. Baioletti, G. Busanello, B. Vantaggi, Conditional independence structure and its closure: inferential rules and algorithms, International Journal of Approximate Reasoning 50 (2009) 1097–1114.
[3] M. Baioletti, G. Busanello, B. Vantaggi, Acyclic directed graphs to represent conditional independence models, in: Lecture Notes ECSQARU 2009, Springer-Verlag, Berlin, Heidelberg, Verona, 2009, pp. 530–541.
[4] R.R. Bouckaert, M. Studený, Racing algorithms for conditional independence inference, International Journal of Approximate Reasoning 45 (2007) 386–401.
[5] G. Coletti, R. Scozzafava, Probabilistic Logic in a Coherent Setting, Kluwer, Dordrecht/Boston/London, 2002. Trends in Logic No. 15.
[6] G. Coletti, B. Vantaggi, Possibility theory: conditional independence, Fuzzy Sets and Systems 157 (2006) 1491–1513.
[7] B.R. Cobb, P.P. Shenoy, Operations for inference in continuous Bayesian networks with linear deterministic variables, International Journal of Approximate Reasoning 42 (2006) 21–36.
[8] R.G. Cowell, A.P. Dawid, S.L. Lauritzen, D.J. Spiegelhalter, Probabilistic Networks and Expert Systems, Springer-Verlag, New York, 1999.
[9] F.G. Cozman, T. Seidenfeld, Independence for full conditional measures and their graphoid properties, in: Benedikt Löwe, Eric Pacuit, Jan-Willem Romeijn (Eds.), Foundations of the Formal Sciences VI, Reasoning about Probabilities and Probabilistic Reasoning, Studies in Logic, vol. 16, College Publications, London, 2009, pp. 1–29.
[10] F.G. Cozman, P. Walley, Graphoid properties of epistemic irrelevance and independence, Annals of Mathematics and Artificial Intelligence 45 (2005) 173–195.
[11] D.R. Cox, N. Wermuth, Multivariate Dependencies – Models, Analysis, and Interpretation, Chapman and Hall, 1996.
[12] A.P. Dawid, Conditional independence in statistical theory, Journal of the Royal Statistical Society Series B – Methodological 41 (1979) 15–31.
[13] L.M. de Campos, J.G. Castellano, Bayesian network learning algorithms using structural restrictions international, Journal of Approximate Reasoning 45 (2007) 233–254.
[14] L.M. de Campos, J.F. Huete, Independence concepts in possibility theory I, Fuzzy Sets and Systems 103 (1999) 127–152.
[15] P.R. de Waal, L.C. van der Gaag, Stable independence in perfect maps, in: Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence, UAI'05, Edinburgh, 2005, pp. 161–168.
[16] D. Dor, M. Tarsi, A simple algorithm to construct consistent extension of a partial oriented graph. Technical Report R-185, Cognitive Systems Laboratory, UCLA Computer Science Department, 1992.
[17] A. Feelders, L.C. van der Gaag, Learning Bayesian network parameters under order constraints, International Journal of Approximate Reasoning 42 (2006) 37–53.
[18] D. Geiger, Graphoids: a qualitative framework for probabilistic inference, PhD Thesis, University of California, Los Angeles, 1990.
[19] F.V. Jensen, An Introduction to Bayesian Networks, UCL Press, Springer-Verlag, 1996.
[20] R. Jirousek, J. Vejnarová, Compositional models and conditional independence in evidence theory. International Journal of Approximate Reasoning, in press, doi:10.1016/j.ijar.2010.02.005.
[21] S.L. Lauritzen, Graphical Models, Clarendon Press, Oxford, 1996.
[22] S. Moral, A. Cano, Strong conditional independence for credal sets, Annals of Mathematics and Artificial Intelligence 35 (2002) 295–321.
[23] M. Niepert, D. Van Gucht, M. Gyssens, Logical and algorithmic properties of stable conditional independence, International Journal of Approximate Reasoning 51 (5) (2010) 531–543.
[24] A. Paz, Annotated graphs model for representing DAG – Representable relations. Algorithmic approach and extensions, in: J. Vejnarová (Ed.), Proceedings Sixth Workshop on Uncertainty, WUPES 2003, 2003, pp. 205–220.
[25] A. Paz, R.Y. Geva, M. Studený, Representation of irrelevance relations by annonated graphs, Fundamentae Informaticae 34 (2000) 1–51.
[26] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann, Los Altos, CA, 1988.
[27] J. Pearl, Causality, Cambridge University Press, 2000.
[28] J. Pearl, A. Paz,. Graphoids: a graph-based logic for reasoning about relevance relations, Technical Report 850038 (R-52), UCLA Computer Science Department, 1985.
[29] J. Pearl, T. Verma, Equivalence and synthesis of causal models, in: Sixth Conference in Uncertainty in Artificial Intelligence, UAI '90, 1990, pp. 255–270. Also in Uncertainty in Artificial Intelligence, Elsevier Science Publishers, Cambridge, MA, 1991, pp. 220–227.
[30] J. Pearl, T. Verma, An algorithm for deciding if a set of observed independencies has a causal explanation, in: Eighth Conference in Uncertainty in Artificial Intelligence, UAI'92, 1992, pp. 323–330.
[31] P.P. Shenoy, Conditional independence in valuation-based systems, International Journal of Approximate Reasoning 10 (1994) 203–234.
[32] P. Spirtes, C. Glymour, R. Scheines, Causation, Prediction, and Search, MIT Press, 2001.
[33] M. Studený, Conditional independence relations have no finite complete characterization, in: S. Kubik, J.A. Visek (Eds.), Information Theory, Statistical Decision Functions and Random Processes Transactions of the 11th Prague Conference, vol. B, Kluwer, Dordrecht, Boston, London, Academia, Prague, 1992, pp. 377–396.
[34] M. Studený, Semigraphoids and structures of probabilistic conditional independence, Annals of Mathematics and Artificial Intelligence 21 (1997) 71–98.
[35] M. Studený, Complexity of structural models, in: Proceedings of the Prague Stochastics'98, Prague, 1998, pp. 521–528.
[36] M. Studený, R.R. Bouckaert, On chain graph models for description of conditional independence structures, Annals of Statistics 26 (4) (1998) 1434–1495.
[37] M. Studený, Probabilistic Conditional Independence Structures, Springer-Verlag, London, 2005.
[38] B. Vantaggi, Conditional independence in a coherent setting, Annals of Mathematics and Artificial Intelligence 32 (2001) 287–313.
[39] B. Vantaggi, The L-separation criterion for description of cs-independence models, International Journal of Approximate Reasoning 29 (3) (2002) 291–316.
[40] B. Vantaggi, Conditional independence structures and graphical models, International Journal of Uncertainty, Fuzziness Knowledge-Based Systems 11 (5) (2003) 545–571.
[41] T.S. Verma (1986). Causal networks: semantics and expressiveness. Technical Report R-65, Cognitive Systems Laboratory, University of California, Los Angeles.
[42] J. Veinarová, Conditional independence relations in possibility theory, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 8 (3) (2000) 253–269.

[43] T.S. Verma, J. Pearl, Equivalence and synthesis of causal models, in: Uncertainty in Artificial Intelligence, vol. 6, 1991, pp. 220–227.
[44] J. Witthaker, Graphical Models in Applied Multivariate Statistic, Wiley and Sons, New York, 1990.
[45] S.K.M. Wong, C.J. Butz, D. Wu, On the implication problem for probabilistic conditional independency, IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans 30 (6) (2000) 785–805.