International Journal of Approximate Reasoning 126 (2020) 272–289

Contents lists available at ScienceDirect

International Journal of Approximate Reasoning

www.elsevier.com/locate/ijar

A lattice-based representation of independence relations for efficient closure computation

Linda C. van der Gaag [a,b,*], Marco Baioletti [c], Janneke H. Bolt [b]

[a] Dalle Molle Institute for Artificial Intelligence Research, Lugano, Switzerland
[b] Department of Information and Computing Sciences, Utrecht University, the Netherlands
[c] Dipartimento di Matematica e Informatica, Università degli Studi di Perugia, Italy

ARTICLE INFO

Article history:
Received 30 March 2019
Received in revised form 7 August 2020
Accepted 9 August 2020
Available online 28 August 2020

Keywords:
Independence relations
Representation
Lattice-based partitioning
Fast-closure computation
Algorithm engineering

ABSTRACT

Independence relations in general include exponentially many statements of independence, that is, exponential in the number of variables involved. These relations are typically characterised however, by a small set of such statements and an associated set of derivation rules. While various computational problems on independence relations can be solved by manipulating these smaller sets without the need to explicitly generate the full relation, existing algorithms for constructing these sets are associated with often prohibitively high running times. In this paper, we introduce a lattice structure for organising sets of independence statements and show that current algorithms are rendered computationally less demanding by exploiting new insights in the structural properties of independence gained from this lattice organisation. By means of a range of experimental results, we subsequently demonstrate that through the lattice organisation indeed a substantial gain in efficiency is achieved for fast-closure computation of semi-graphoid independence relations in practice.

1. Introduction

Probabilistic independence is key to the scalability of probabilistic models, as is demonstrated by the practicability of probabilistic graphical models encoding such independences (see for example [4,6]). Probabilistic independence therefore is a subject of intensive studies from both a mathematics and a computing-science perspective (see [5,8,12]). Pearl and his co-workers were among the first to formalise properties of probabilistic independence in a system of axioms [8]. These axioms are now commonly taken as derivation rules for generating new independences from a given set of independence statements. Any set of such statements that is closed under finite application of these rules is called an *independence relation*.

Independence relations in general are exponentially large in the number of variables involved. Representing them by enumeration of their individual independence statements therefore is not feasible for practical purposes. These relations are typically fully characterised however, by a much smaller set of statements, called a *basis*, and an associated set of derivation rules. Studený was the first to propose a type of basis that allows various computational problems on independence relations to be solved efficiently without the need to generate the full relation [10,11]. He designed an elegant algorithm for computing such a basis from a given starting set of independence statements, which was later improved upon [2,7] to yield the current state-of-the-art algorithm for *fast-closure computation*.

* Corresponding author.
E-mail addresses: linda.vandergaag@idsia.ch (L.C. van der Gaag), marco.baioletti@unipg.it (M. Baioletti), j.h.bolt@uu.nl (J.H. Bolt).

https://doi.org/10.1016/j.ijar.2020.08.002
0888-613X/

While Studený's basis allows efficiently solving a range of problems on independence relations, existing algorithms for constructing such a basis are still highly demanding from a computational perspective. In this paper, we introduce a lattice structure for organising sets of independence statements which aims at reducing the running time of the state-of-the-art algorithm for fast-closure computation by means of a partitioning approach. Although the proposed lattice representation accommodates any type of independence relation, we will focus the discussion of its properties on the class of semi-graphoid independence relations. We will show that the organisational lattice structure supports efficient maintenance of a non-redundant set of independence statements, which is one of the most demanding steps of fast-closure computation, by restricting redundancy checks to parts of the lattice; the organisational structure further allows efficient selection of independence statements for application of the main operator involved in closure computation. Our experimental results from fast-closure computation with up to 80 variables demonstrate that the computational advantages obtained are substantial.

The paper is organised as follows. In Section 2, we review some basic notions from semi-graphoid independence relations and fast-closure computation, and thereby introduce our notations. In Section 3, our lattice structure for organising sets of independence statements is introduced and some of its properties are stated. Section 4 then discusses the computational advantages, for fast-closure computation, of the new representation. Section 5 describes our experiments and details the results obtained. The paper ends in Section 6 with our concluding observations and plans for further research.

2. Preliminaries

We consider a finite, non-empty set V of discrete random variables, with $|V| = n$, $n \geq 2$. We will use capital letters to indicate subsets of this set V. Small letters are used to denote individual variables; when indicating individual variables in a set, we slightly abuse notational conventions and write pqr instead of $\{p, q, r\}$. An ordered *triplet* over V is a statement of the form $\theta = \langle A, B \,|\, C\rangle$, where A, B, C are pairwise disjoint subsets of V with $A, B \neq \varnothing$. A triplet $\langle A, B \,|\, C\rangle$ is taken to state that the sets of variables A and B are (conditionally) independent given the set C. Relative to a discrete joint probability distribution Pr over V, the triplet thus states that the schema $\Pr(A, B \,|\, C) = \Pr(A \,|\, C) \cdot \Pr(B \,|\, C)$ holds for all joint value combinations of $A \cup B \cup C$. In the sequel, we will use $X = A \cup B \cup C$ to denote the set of all variables involved in the triplet θ; the set C of the triplet will be referred to as the triplet's *separating set*. Any triplet with $X = V$ is called a *saturated triplet*. The *symmetric transpose* of a triplet $\theta = \langle A, B \,|\, C\rangle$, denoted as θ^T, is the triplet $\langle B, A \,|\, C\rangle$; we note that, as triplets are taken to be ordered, a triplet θ and its transpose are considered different statements of independence. The set of all possible triplets over V is denoted as $V^{(3)}$.

A set of triplets constitutes a *semi-graphoid independence relation* if it satisfies the four properties stated in the following definition.

Definition 1. A *semi-graphoid independence relation* is a set of triplets $\overline{J} \subseteq V^{(3)}$ that satisfies the following properties:

G1: if $\langle A, B \mid C\rangle \in \overline{J}$, then $\langle B, A \mid C\rangle \in \overline{J}$ *(Symmetry)*
G2: if $\langle A, B \mid C\rangle \in \overline{J}$, then $\langle A, B' \mid C\rangle \in \overline{J}$ for any non-empty subset $B' \subseteq B$ *(Decomposition)*
G3: if $\langle A, B_1 \cup B_2 \mid C\rangle \in \overline{J}$ with B_1 non-empty and $B_1 \cap B_2 = \varnothing$, then $\langle A, B_1 \mid C \cup B_2\rangle \in \overline{J}$ *(Weak Union)*
G4: if $\langle A, B \mid C \cup D\rangle \in \overline{J}$ and $\langle A, C \mid D\rangle \in \overline{J}$, then $\langle A, B \cup C \mid D\rangle \in \overline{J}$ *(Contraction)*

The four properties stated above constitute an axiomatic system for the qualitative notion of (conditional) independence [8]. This system of axioms is sound for the concept of probabilistic independence in the class of discrete probability distributions [5], yet not complete; it has been shown, in fact, that the probabilistic notion of independence does not allow a finite axiomatisation [9].

The semi-graphoid properties of independence G1–G4 are typically taken as derivation rules for generating (possibly) new triplets from a given triplet set. Given a *starting set* of triplets $J \subseteq V^{(3)}$ and a designated triplet $\theta \in V^{(3)}$, we write $J \vdash^* \theta$ if the triplet θ can be derived from J by finite application of the four semi-graphoid rules. The *full closure* of a starting triplet set $J \subseteq V^{(3)}$, denoted as $\overline{J}$, is the semi-graphoid independence relation consisting of J and all triplets θ that can be derived from it. This full closure typically is exponentially large in the number of variables in the set V. For concisely representing a semi-graphoid independence relation, it now suffices to find an appropriate subset of triplets, called a *basis*, that captures, jointly with the four derivation rules, the same information as the full relation itself.

Definition 2. Let $\overline{J} \subseteq V^{(3)}$ be a semi-graphoid independence relation. Then, any triplet set $J' \subseteq V^{(3)}$ with $\overline{J'} = \overline{J}$ is called a *basis* for $\overline{J}$.

Many computational problems on independence relations can be solved from a starting set of triplets J directly, without the need to first generate its full closure. Studený was the first to design an algorithm for constructing, from an arbitrary starting set of triplets, a tailored type of basis for semi-graphoid independence relations [11], which later became known as the *fast-closure basis* [2]. From a fast-closure basis, for example, the *implication problem* for semi-graphoid relations can be solved efficiently. This problem asks for a given starting set J and triplet θ whether $J \vdash^* \theta$, and is solvable in a time linear in the size of the fast-closure basis of the starting set J at hand. In this paper, we focus on the computation of such fast-closure bases.

We briefly review the various notions underlying Studený's fast-closure basis. To this end, we begin with observing that the semi-graphoid rules G2–G3 define a *derivational partial order* among triplets which was formalised by Studený in the notion of *dominance* [11]. This notion was later enhanced by including the symmetry rule G1, to the notion of *g-inclusion* by Baioletti et al. [2]; from now on, we build on the latter notion.

Definition 3. Let $J \subseteq V^{(3)}$ be an arbitrary set of triplets and let $\theta_i = \langle A_i, B_i \,|\, C_i \rangle \in J$, with $X_i = A_i \cup B_i \cup C_i$, $i = 1, 2$. Then, θ_1 is *g-included* in θ_2, denoted as $\theta_1 \sqsubseteq \theta_2$, if the following conditions hold:

- $C_2 \subseteq C_1 \subseteq X_2$; and,
- $[A_1 \subseteq A_2$ and $B_1 \subseteq B_2]$, or $[B_1 \subseteq A_2$ and $A_1 \subseteq B_2]$.

A triplet $\theta \in J$ is *g-maximal* in J if it is not g-included in any triplet $\tau \in J$ with $\tau \neq \theta, \theta^T$.

If a triplet θ_1 is g-included in another triplet θ_2, then we have that any triplet that can be derived from θ_1 by means of the derivation rules G1–G3 can also be derived from θ_2 through these rules. For describing a semi-graphoid independence relation by a subset of its triplets therefore, only g-maximal triplets need be represented explicitly. We note that, if a triplet θ and its symmetric transpose θ^T are both included in a triplet set J, then g-maximality of θ in J implies g-maximality of θ^T in J, and vice versa. Since θ can be derived from θ^T and vice versa, just one of this pair of symmetric triplets suffices for capturing all information from the full relation. A set of g-maximal triplets which includes at most one of each pair of symmetric triplets is termed a *non-redundant triplet set*.

Definition 4. Let $J \subseteq V^{(3)}$ be an arbitrary set of triplets. Then, J is *non-redundant* if the following conditions hold:

- $\theta \in J$ implies $\theta^T \notin J$; and,
- each $\theta \in J$ is g-maximal in J.

Thus far, we addressed derivability among triplets in view of just the derivation rules G1–G3, each of which derives a potentially new triplet from a *single* triplet in a triplet set J. The contraction rule G4 differs from G1–G3 in that it combines information from two triplets for constructing a potentially new one. For application of this rule, Studený formulated a generalised contraction operator, called the *gc-operator* [11], which is defined as follows.

Definition 5. Let V be as before. For all triplets $\theta_i = \langle A_i, B_i \,|\, C_i \rangle \in V^{(3)}$ with $X_i = A_i \cup B_i \cup C_i, i = 1, 2$, such that

- $A_1 \cap A_2 \neq \varnothing$;
- $C_1 \subseteq X_2$ and $C_2 \subseteq X_1$; and
- $(B_2 \setminus C_1) \cup (B_1 \cap X_2) \neq \varnothing$;

the *gc-operator* is defined through:

$$gc(\theta_1, \theta_2) = \langle A_1 \cap A_2,\ (B_2 \setminus C_1) \cup (B_1 \cap X_2) \ \mid\ C_1 \cup (A_1 \cap C_2) \rangle$$

For any triplet pair θ_1, θ_2 for which the above conditions do not all hold, $gc(\theta_1, \theta_2)$ is undefined.

The *gc*-operator in essence constructs from two triplets θ_1, θ_2, two g-included triplets θ_1', θ_2' with $\theta_1' \sqsubseteq \theta_1$ and $\theta_2' \sqsubseteq \theta_2$, to which the derivation rule G4 can be applied to yield a possibly new g-maximal triplet. We note that by considering not just the triplets θ_1, θ_2 but also all their g-included triplets, it suffices to maintain a non-redundant triplet set for iteratively applying the operator. Studený further showed that if $gc(\theta_1, \theta_2)$ yields a valid triplet θ, then this triplet g-includes any other triplet that can be obtained from applying the derivation rule G4 to triplets derived from θ_1, θ_2 through G1–G3 [11]; the *gc*-operator is thus guaranteed to yield the highest-ordered triplet of the set of all triplets derivable from the ordered pair θ_1, θ_2.

We illustrate the basic idea of the *gc*-operator by means of a small example.

Example 1. We consider the two triplets

$$\theta_1 = \langle p, q \,|\, r \rangle,\ \theta_2 = \langle p, rs \,|\, q \rangle$$

over the variable set $V = \{p, q, r, s\}$, and address application of the derivation rule G4. When applied to the pair θ_1, θ_2 directly, the rule does not yield a new triplet since the separating sets of the two triplets are not empty yet have an empty intersection. We now observe that from the triplet θ_2 the triplet $\theta_2' = \langle p, s \,|\, qr \rangle$ can be derived by using the derivation rule G3. G4 can then be applied to the pair θ_1, θ_2' to give the new triplet $\theta = \langle p, qs \,|\, r \rangle$. Application of the *gc*-operator to the original triplet pair θ_1, θ_2 yields $gc(\theta_1, \theta_2) = \theta$ directly, and hence forestalls the necessity of explicitly maintaining or deriving θ_2'.

FAST CLOSURE CONSTRUCTION:
Input: $J \subseteq V^{(3)}$; *Output*: F with $\overline{F} = \overline{J}$

1: **function** *Basis-Construction*(J)
2: $J_0 \leftarrow J$;
3: $i \leftarrow 0$;
4: **repeat**
5: $i \leftarrow i + 1$;
6: $N_i \leftarrow \bigcup_{\theta_1, \theta_2 \in J_{i-1}} (GC(\theta_1, \theta_2) \cup GC(\theta_2, \theta_1))$;
7: $J_i \leftarrow$ *non-redundant*($J_{i-1} \cup N_i$)
8: **until** $J_i = J_{i-1}$;
9: **return** $F \leftarrow J_i$
10: **end function**

Algorithm 1. A sketch of the state-of-the-art algorithm for fast-closure computation (adapted from [7]).

In view of a given triplet set, when an ordered pair of triplets θ_1, θ_2 is selected for application of the *gc*-operator, typically not just $gc(\theta_1, \theta_2)$ but also $gc(\theta_1^T, \theta_2)$, $gc(\theta_1, \theta_2^T)$ and $gc(\theta_1^T, \theta_2^T)$ are established, before selecting the next triplet pair. In the sequel, we will use $GC(\theta_1, \theta_2)$ to denote the set of all valid triplets resulting from such a fourfold application of the operator. We note that for any triplet pair θ_1, θ_2, the set $GC(\theta_1, \theta_2) \cup GC(\theta_2, \theta_1)$ may include up to eight valid triplets.

Building upon the concepts reviewed above, Studený designed an algorithm for constructing, from a starting set of triplets, a fast-closure basis composed of maximal triplets. This algorithm was enhanced, first by Baioletti et al. [2] and later by Lopatatzidis and Van der Gaag [7], to the current state-of-the-art *algorithm for fast-closure computation*; Algorithm 1 provides a high-level sketch of this algorithm. The core of the algorithm is an iterative loop (lines 4–8) in which new triplets are derived and redundant triplets are removed. More specifically, in each iteration, starting with a triplet set J_{i-1}, the algorithm computes the set N_i of triplets that result from application of the *GC*-operator to all possible (ordered) pairs of triplets from J_{i-1} (line 6); from the joined set $J_{i-1} \cup N_i$ then all redundant triplets are removed (line 7). With the resulting non-redundant triplet set J_i, the loop is repeated until the set J_i no longer changes, in the sense that no new g-maximal triplets are added. The returned triplet set F then is a fast-closure basis for the independence relation $\overline{J}$ defined by the original starting set.

The algorithm is generally thought to have a worst-case runtime complexity that is exponential in the size of the starting triplet set J_0. In each iteration of the algorithm's main loop, line 6 takes a runtime that is polynomial in the size of this set. With $|J_0| = d$, the first iteration of line 6 takes at most $O(d^2)$ time. In the worst case, this iteration may result in a set N_1 with $O(d^2)$ new triplets. In general, the ith iteration of line 6 may take $O(d^{2^i})$ time and result in a triplet set of a size up to $O(d^{2^i})$. While each iteration thus takes polynomial time in d, the power of the polynomial involved is exponential in the iteration number. Similar observations hold for the worst-case runtime per iteration over line 7. The overall runtime of the algorithm is dependent not just of the running times taken by lines 6 and 7 per iteration however, but also of the number of iterations. The number of iterations required before meeting the stopping criterion in line 8 for the algorithm's main loop, may in fact be exponential in the size of the starting triplet set, as the algorithm may need to reconstruct a sizeable part of the full closure of this set. To the best of our knowledge, the exact relation between the number of iterations required and the sizes of the intermediate triplet sets constructed in these iterations is still an open question.

3. An organisational lattice structure for triplet sets

We introduce our lattice structure for organising arbitrary sets of triplets by their separating sets, and study the structural properties of triplet sets as induced by these separating sets.

3.1. Partitioning a triplet set

We consider the power set of the set V of variables, with $|V| = n$, $n \geq 2$, and its representation by means of a lattice $\mathcal{L}$. As usual, each element of the lattice corresponds to a subset of V and the partial ordering among the lattice's elements corresponds to the inclusion relation among these subsets. The element $\varnothing$ constitutes the *least element* of the lattice, and its *greatest element* is the full set V. In the sequel, we will also refer to (*closed*) *intervals* of the lattice $\mathcal{L}$, and write $\mathcal{L}([W; Z])$ with $W \subseteq Z \subseteq V$ to denote the interval of $\mathcal{L}$ with W for its least element and Z for its greatest element; we use $\mathcal{L}(C)$ to indicate the single element corresponding to the subset C of V. For lattice concepts and terminology, we refer the reader to [3].

We now take the lattice $\mathcal{L}$ as an organisational data structure for storing a set J of triplets over V, by looking upon $\mathcal{L}$'s elements as representing *separating sets*: at the element $\mathcal{L}(C)$ will be stored the subset of all triplets of J that have C for their separating set. For a given triplet set J, we will write $\mathcal{L}_J(C) = \{\theta \mid \theta = \langle A, B \mid C \rangle \in J\}$ to denote the set of triplets stored at the lattice element $\mathcal{L}(C)$ after organising J over $\mathcal{L}$. We note that, since the separating set of a triplet cannot include more than $n - 2$ variables, the semantics assigned to the lattice structure does not allow any triplet to be

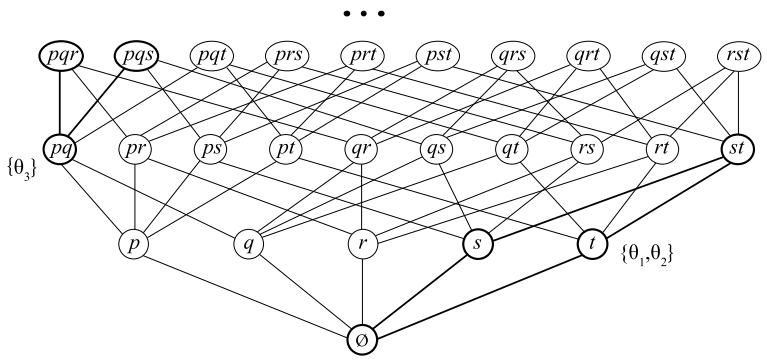

Fig. 1. The lattice $\mathcal{L}$ for the variable set $V = \{p, q, r, s, t\}$, with a set of three triplets organised over it; the bold lines indicate the two intervals $\mathcal{L}([\varnothing; st])$ and $\mathcal{L}([pq; pqrs])$ (the necessarily empty element of the latter interval is not shown).

stored at the two upper levels of the lattice. For any triplet set J, we thus have that $\mathcal{L}_J(W) = \varnothing$ for all subsets $W \subseteq V$ with $|W| \geq n - 1$. For ease of presentation, we will retain, throughout the paper, the perspective of a full lattice for our organisational data structure; for practical implementations however, a meet lattice, without the two upper levels, would be employed.

Using the organisational lattice structure described above, a triplet set J is partitioned into mutually disjoint subsets stored at different elements of the lattice. We illustrate the basic idea with a small example.

Example 2. We consider the set of variables $V = \{p, q, r, s, t\}$. The four levels of the lattice $\mathcal{L}$ over V at which triplets can be stored, are shown in Fig. 1; the upper two levels of the full lattice over V have been replaced by dots to indicate that these levels cannot be used for storing triplets over V. In the figure, the intervals $\mathcal{L}([\varnothing; st])$ and $\mathcal{L}([pq; pqrs])$ are highlighted by bold lines; of the latter interval, the element at which no triplets can be stored by the semantics of the lattice, is not shown. We now consider the set J composed of the three triplets

$$\theta_1 = \langle p, q \mid t \rangle,\ \theta_2 = \langle q, rs \mid t \rangle,\ \textit{and}\ \theta_3 = \langle r, st \mid pq \rangle$$

and organise this set over the lattice structure $\mathcal{L}$. We find that $\mathcal{L}_J(t) = \{\theta_1, \theta_2\}$ and $\mathcal{L}_J(pq) = \{\theta_3\}$; all other lattice elements remain empty.

To gain insight in how triplet sets are partitioned over the organisational lattice structure, we investigate the cardinalities of the subsets of triplets that can maximally be stored at the different elements of the lattice. The following lemma states the maximum number of triplets per lattice element.

Lemma 1. *Let V be a set of variables, with $|V| = n$, $n \geq 2$. Then, for each $C \subseteq V$ with $|C| = n - m$, $n \geq m \geq 2$, there exist at most $3^m - 2^{m+1} + 1$ different triplets with C for their separating set.*

Proof. We consider triplets of the form $\langle A_j, B_j | C \rangle$ with a fixed separating set C of cardinality $|C| = n - m$. With $|V \setminus C| = m$, the remaining m variables may be included in the set $A_j \cup B_j$ of such a triplet. For any fixed selection of $i \geq 2$ variables from among these m variables, there exist $S(i, 2) = 2^{i-1} - 1$ partitions into two non-empty subsets A_j, B_j, where $S(i, 2)$ is the associated Stirling partition number. Each of these partitions defines an *unordered* pair of variable sets, and therefore has associated two symmetric triplets in which the two sets are ordered. As there are $\binom{m}{i}$ ways to choose exactly i variables from the set of m, there are at most $2 \cdot \binom{m}{i} \cdot (2^{i-1} - 1)$ triplets of the form $\langle A_j, B_j \mid C \rangle$ with $|A_j \cup B_j| = i$. By summing over all values $i = 2, \ldots, m$, the number of different triplets with C for their separating set is found to be at most

$$2 \cdot \sum_{i=2}^{m} \binom{m}{i} \cdot (2^{i-1} - 1) \;=\; \sum_{i=0}^{m} \binom{m}{i} \cdot (2^i - 2) + 1 \;=\; 3^m - 2^{m+1} + 1$$

as stated in the lemma. □

The number stated in Lemma 1 is the maximum number of triplets that can be stored at a single element of the organisational lattice structure, that is, the number indicates the maximum cardinality of the set $\mathcal{L}_J(C)$, taken over all possible triplet sets $J \subseteq V^{(3)}$. From this number of different triplets per separating set, we readily derive the maximum number of triplets that a semi-graphoid independence relation can include.

Corollary 2. *Any triplet set $J \subseteq V^{(3)}$ includes at most $4^n - 2 \cdot 3^n + 2^n$ different triplets.*

Proof. We consider again triplets of the form $\langle A_j, B_j \,|\, C\rangle$, yet now for all possible separating sets C with cardinalities $|C| = 0, \ldots, n-2$. The number m of variables that may be included in the set $A_j \cup B_j$ of a triplet, then equals $m = 2, \ldots, n$, respectively. Using the property stated in Lemma 1, the maximum number of different triplets is found to be

$$\sum_{m=2}^{n} \binom{n}{m} \cdot (3^m - 2^{m+1} + 1) \;=\; 4^n - 2 \cdot 3^n + 2^n$$

as stated. □

The number stated in the corollary above is the maximum number of triplets that can be constructed over V, and thus equals the cardinality of $V^{(3)}$. The triplet set that includes this maximum number of triplets, constitutes a full semi-graphoid independence relation in which every pair of sets of variables is independent given every possible separating set.

The maximum numbers of different triplets stated in Lemma 1 and Corollary 2, are found only with a full semi-graphoid independence relation and, hence, for a triplet set that includes all associated symmetric transposes and g-included triplets. The number of different triplets in the largest possible *non-redundant* triplet set for a fixed separating set, is considerably smaller than the number stated in Lemma 1. By observing that a non-redundant triplet set does not include any symmetric transposes, we know that the size of this largest possible set is smaller than half the number from the lemma, that is, smaller than $\frac{3^m+1}{2} - 2^m$. In view of this upper bound on the cardinality of the largest possible non-redundant triplet set for a fixed separating set, we now derive a lower bound on this cardinality. To this end, we focus on triplet sets without any transposes, that is, we consider the first two arguments of a triplet as being *unordered*. For such sets of unordered triplets, we introduce the concept of *stratification*.

Definition 6. Let $J_C \subseteq V^{(3)}$, $|V| = n$, $n \geq 2$, be a set of unordered triplets of the form $\langle A, B \,|\, C\rangle$ where C is a fixed separating set with $|C| = n - m$, $n \geq m \geq 2$, and $A, B \subseteq V \setminus C$, $A \cap B = \varnothing$. The *stratification* of J_C is the partition $\left[J_C^2, \ldots, J_C^m\right]$ of J_C in which $J_C^i = \{\theta \mid \theta = \langle A, B \,|\, C\rangle \in J_C,\ |A \cup B| = i\}$, $i = 2, \ldots, m$. Each block J_C^i of the partition is called a *stratum* of the stratification of J_C.

We note that a stratum J_C^i of the stratification of a set J_C of unordered triplets, includes all triplets from J_C with exactly i variables in their first two arguments jointly. The maximum cardinality of the stratum J_C^i is derived by a similar argument as used for Lemma 1, and equals $\binom{m}{i} \cdot (2^{i-1} - 1)$. We illustrate the basic idea of the stratification of a triplet set with a fixed separating set, by means of an example.

Example 3. We consider the set of variables $V = \{p, q, r, s, t\}$, and the separating set $C = \varnothing$; we thus have that $n = m = 5$. Any set $J_\varnothing$ of unordered triplets stored at the lattice element $\mathcal{L}(\varnothing)$ has a stratification $\left[J_\varnothing^2, J_\varnothing^3, J_\varnothing^4, J_\varnothing^5\right]$ consisting of four strata. The stratum $J_\varnothing^2$, for example, can include at most ten triplets, with the following pairs of singleton sets for their first two arguments:

$$\begin{array}{llllll} (p,q), & (p,s), & (q,r), & (q,t), & (r,t), \\ (p,r), & (p,t), & (q,s), & (r,s), & (s,t). \end{array}$$

The four strata of the stratification of $J_\varnothing$ have the following maximum cardinalities, respectively:

$$|J_\varnothing^2| = \binom{5}{2} \cdot (2^1 - 1) = 10 \qquad |J_\varnothing^4| = \binom{5}{4} \cdot (2^3 - 1) = 35$$
$$|J_\varnothing^3| = \binom{5}{3} \cdot (2^2 - 1) = 30 \qquad |J_\varnothing^5| = \binom{5}{5} \cdot (2^4 - 1) = 15$$

Among these four strata, the stratum $J_\varnothing^4$ can include the largest number of triplets.

To establish a lower bound on the size of the largest possible non-redundant triplet set that can be stored at the lattice element $\mathcal{L}(C)$, we now first show that, for any number i, the stratum J_C^i of the stratification of a set J_C of unordered triplets constitutes a non-redundant triplet set. We will shortly build upon this property to show that the maximum size of the stratum that can include the largest number of triplets, is a lower bound on the cardinality of the largest possible non-redundant triplet set that can be stored at $\mathcal{L}(C)$.

Proposition 3. *Let $J_C \subseteq V^{(3)}$, $|V| = n$, $n \geq 2$, be a set of unordered triplets with a fixed separating set C with $|C| = n - m$, $n \geq m \geq 2$. Then, for all $i = 2, \ldots, m$, the stratum J_C^i of the stratification of J_C constitutes a non-redundant triplet set.*

Proof. We have to show that $\theta_1 \not\sqsubseteq \theta_2$ for any ordered pair of different triplets θ_1, θ_2 from the stratum J_C^i for some i. Since by the construction of J_C^i we have that $|A_1 \cup B_1| = |A_2 \cup B_2|$, we find for all θ_1, θ_2 with $|A_1| \neq |A_2|$ that the conditions

$A_1 \subseteq A_2$ and $B_1 \subseteq B_2$ cannot hold simultaneously. For all θ_1, θ_2 with $|A_1| = |A_2|$ and, hence, $|B_1| = |B_2|$, moreover, we must have that if $A_1 = A_2$ then $B_1 \neq B_2$ and vice versa, from which we equally have that the conditions $A_1 \subseteq A_2$ and $B_1 \subseteq B_2$ cannot both hold. As similar observations hold with respect to θ_1^T, we have that $\theta_1 \not\sqsubseteq \theta_2$ for all ordered pairs $\theta_1, \theta_2 \in J_C^i$. We conclude that J_C^i is a non-redundant triplet set. □

From Proposition 3 we have that each stratum J_C^i, $i = 2, \ldots, m$, of *any* set of unordered triplets J_C constitutes a non-redundant triplet set. We now consider the largest possible triplet set K_C consisting of all unordered triplets with the separating set C, and argue that the cardinality of the largest stratum of this set K_C constitutes a lower bound on the size of the largest possible *non-redundant* triplet set within K_C. The following proposition characterises this stratum of maximum cardinality among the strata of the triplet set K_C.

Proposition 4. *Let $K_C \subseteq V^{(3)}$, $|V| = n$, $n \geq 2$, be the largest possible set of unordered triplets with a fixed separating set C with $|C| = n - m$, $n \geq m \geq 2$. Then, for $k = \left\lfloor \frac{2}{3}m + \frac{2}{3} \right\rfloor$, we have that $|K_C^k| \geq |K_C^i|$ for all $i = 2, \ldots, m$.*

Proof. Since the property stated in the lemma is readily verified to hold for $m = 2$, $m = 3$, we assume for the remainder of the proof that $m \geq 4$.

We consider two consecutive strata K_C^i, K_C^{i+1}, $i = 2, \ldots, m-1$, of the stratification of the set K_C. For these strata, we have that $|K_C^i| = \binom{m}{i} \cdot (2^{i-1} - 1)$ and $|K_C^{i+1}| = \binom{m}{i+1} \cdot (2^i - 1)$, and hence that $|K_C^{i+1}| = f_m(i) \cdot |K_C^i|$ with

$$f_m(i) = \left(\frac{m-i}{i+1}\right) \cdot \left(2 + \frac{1}{2^{i-1}-1}\right)$$

for $i = 2, \ldots, m-1$. For any number of variables $m \geq 4$, the function $f_m(i)$ decreases strictly monotonically in i:

$$f_m(i) = \left(\frac{m-i}{i+1}\right) \cdot \left(2 + \frac{1}{2^{i-1}-1}\right) > \left(\frac{m-i}{i+1}\right) \cdot \left(2 + \frac{1}{2^{i}-1}\right) > \left(\frac{m-i-1}{i+2}\right) \cdot \left(2 + \frac{1}{2^{i}-1}\right) = f_m(i+1)$$

Since at the extremes of the range of values of i we find that

$$f_m(2) = m - 2 > 1 \quad \text{and} \quad f_m(m-1) = \frac{1}{m} \cdot \left(2 + \frac{1}{2^{m-2}-1}\right) < 1$$

we are guaranteed that there is a value of i such that $f_m(i-1) \geq 1$ and $f_m(i) < 1$. We now show that there exists a value k at which $f_m(k-1) > 1$ and $f_m(k) < 1$, and that this value equals $k = \left\lfloor \frac{2}{3}m + \frac{2}{3} \right\rfloor$ for all $m \geq 4$. For this purpose, we will distinguish between the three possible forms of m:

- m with $(m \bmod 3) = 0$ has associated $k = \lfloor \frac{2}{3}m + \frac{2}{3} \rfloor = \frac{2}{3}m$ and, hence, $m = \frac{3}{2}k$; since we assumed that $m \geq 4$, the value $m = 6$ is the smallest possible value of m of this form, for which we thus have $k = 4$;
- m with $(m \bmod 3) = 1$ has associated $k = \frac{2}{3}m + \frac{1}{3}$ and, hence, $m = \frac{3}{2}k - \frac{1}{2}$; the smallest possible value of m of this form is $m = 4$, for which we have $k = 3$;
- m with $(m \bmod 3) = 2$ has $k = \frac{2}{3}m + \frac{2}{3}$ and, hence, $m = \frac{3}{2}k - 1$; the smallest possible value of m of this form is $m = 5$, for which we have $k = 4$.

We write $f_m^j(i)$ to indicate the three forms of the function $f_m(i)$, associated with the remainder $j = 0, 1, 2$ after taking modulus 3 of m. To prove that $f_m^j(k-1) > 1$ for $j = 0, 1, 2$, we detail the function values $f_m^j(k-1)$ for the different forms of m, respectively:

- for m with $(m \bmod 3) = 0$, we have that $f_m^0(k-1) = \left(\frac{1}{2} + \frac{1}{k}\right) \cdot \left(2 + \frac{1}{2^{k-2}-1}\right)$;
- for m with $(m \bmod 3) = 1$, we have that $f_m^1(k-1) = \left(\frac{1}{2} + \frac{1}{2k}\right) \cdot \left(2 + \frac{1}{2^{k-2}-1}\right)$;
- for m with $(m \bmod 3) = 2$, we have that $f_m^2(k-1) = \frac{1}{2} \cdot \left(2 + \frac{1}{2^{k-2}-1}\right)$.

The property $f_m^j(k-1) > 1$ for $j = 0, 1, 2$, is now readily seen to hold for all values $k \geq 3$, as for each of the three function values the indicated product expands to 1 plus a positive additional term.

To prove that $f_m^j(k) < 1$ for $j = 0, 1, 2$, we detail the function values $f_m^j(k)$ for the three forms of m distinguished above, respectively:

- for m with $(m \bmod 3) = 0$, we have that $f_m^0(k) = \left(1 - \frac{1}{k+1}\right) \cdot \left(1 + \frac{1}{2^k-2}\right)$;
- for m with $(m \bmod 3) = 1$, we have that $f_m^1(k) = \left(1 - \frac{2}{k+1}\right) \cdot \left(1 + \frac{1}{2^k-2}\right)$;
- for m with $(m \bmod 3) = 2$, we have that $f_m^2(k) = \left(1 - \frac{3}{k+1}\right) \cdot \left(1 + \frac{1}{2^k-2}\right)$.

All three function values $f_m^j(k)$, $j = 0, 1, 2$, expand to $1 + \frac{1}{2^k-2}$ minus a positive term, where the latter term is the smallest for the function value associated with $j = 0$. For this function value $f_m^0(k)$, the positive term equals $\left(\frac{2^k-1}{k+1}\right) \cdot \frac{1}{2^k-2}$. Since for all $k \geq 3$ we have that $\frac{2^k-1}{k+1} > 1$, we find that $f_m^0(k) < 1$. As the positive terms in the function values $f_m^1(k)$, $f_m^2(k)$ are larger than the term $\left(\frac{2^k-1}{k+1}\right) \cdot \frac{1}{2^k-2}$ in $f_m^0(k)$, the properties $f_m^1(k) < 1$, $f_m^2(k) < 1$ readily follow.

From the arguments above, we find that, for all $m \geq 2$, the stratum K_C^k with $k = \lfloor \frac{2}{3}m + \frac{2}{3} \rfloor$ has largest cardinality among all strata K_C^i, $i = 2, \ldots, m$, of the stratification of K_C. □

Propositions 3 and 4 jointly show that the largest non-redundant triplet set that can be stored at the lattice element $\mathcal{L}(C)$, has at least $|K_C^k|$ triplets with $k = \left\lfloor \frac{2}{3}m + \frac{2}{3} \right\rfloor$. Now suppose that there exists in K_C a non-redundant subset of triplets of strictly larger cardinality than $|K_C^k|$. It is easily seen that this subset cannot be found by adding triplets to K_C^k, since

- any triplet from the set $K_C^{k+1} \cup \ldots \cup K_C^m$ g-includes at least one triplet from K_C^k;
- any triplet from the set $K_C^2 \cup \ldots \cup K_C^{k-1}$ is g-included in at least one triplet from K_C^k.

Although this observation does not guarantee that there does not exist in K_C a non-redundant subset of triplets of larger cardinality than $|K_C^k|$, we cautiously conjecture that it is not possible to construct such a non-redundant triplet set of larger cardinality.

Conjecture 5. *Let $J_C \subseteq V^{(3)}$, $|V| = n$, $n \geq 2$, be an arbitrary set of unordered triplets with a fixed separating set C with $|C| = n - m$, $n \geq m \geq 2$. Then, any non-redundant subset of J_C includes at most $|J_C^k|$ triplets, with $k = \lfloor \frac{2}{3}m + \frac{2}{3} \rfloor$.*

If the conjecture would be true, then $\binom{m}{k} \cdot (2^{k-1} - 1)$ is a tight upper bound on the size of any non-redundant triplet set represented at the lattice element $\mathcal{L}(C)$. If, on the other hand, a counterexample would be found, proving the conjecture to be wrong, the maximum size of any non-redundant triplet set represented at $\mathcal{L}(C)$ must lie between $\binom{m}{k} \cdot (2^{k-1} - 1)$ and $\sum_{i=2}^{m} \binom{m}{i} \cdot (2^{i-1} - 1)$, where the latter expression follows from Lemma 1 and the observation that a non-redundant triplet set does not include any transposes.

3.2. Dynamically maintaining the triplet-set organisation

Upon fast-closure computation of a given triplet set, new triplets are derived through application of the *gc*-operator. When the triplet set is partitioned over an organisational lattice structure as described in the previous section, the newly derived triplets have to be inserted at the appropriate elements of this structure. Now, if application of the *gc*-operator to an ordered triplet pair yields a valid triplet, the lattice element at which this possibly new triplet has to be inserted, is determined to a large extent by the separating sets of the two original triplets from it was derived. The following proposition supports this observation.

Proposition 6. *Let V be as before and let $\theta_i = \langle A_i, B_i | C_i \rangle \in V^{(3)}$, $i = 1, 2$, be such that $\theta = gc(\theta_1, \theta_2)$ is a valid triplet with a separating set C. Then, the following properties hold:*

- *if $C_1 \subseteq C_2$, then $C_1 \subseteq C \subseteq C_2$;*
- *if $C_2 \subseteq C_1$, then $C = C_1$;*
- *otherwise, $C_1 \subseteq C \subseteq C_1 \cup C_2$.*

Proof. We prove the first property stated in the proposition. To this end, we observe that the triplet $\theta = gc(\theta_1, \theta_2)$ constructed from θ_1, θ_2, has $C = C_1 \cup (A_1 \cap C_2)$ for its separating set. From the condition $C_1 \subseteq C_2$, we readily find that C has $C_1 \subseteq C \subseteq C_2$, as stated. The second property is proven analogously. To prove the third property stated in the proposition, we observe that the separating set C of the triplet θ includes C_1 and a (possibly empty) subset of $C_2 \setminus C_1$. We thus find that $C_1 \subseteq C \subseteq C_1 \cup C_2$, as stated. □

We briefly address the implications of Proposition 6 for maintaining a set of triplets in the organisational lattice structure upon fast-closure computation. We consider to this end two triplets θ_1, θ_2 taken from lattice elements $\mathcal{L}(C_1), \mathcal{L}(C_2)$,

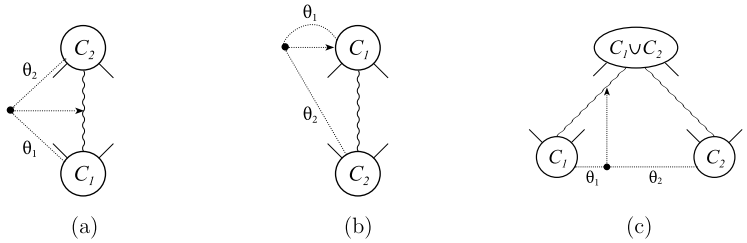

Fig. 2. The dynamics of inserting a valid triplet $gc(\theta_1, \theta_2)$ into the organisational lattice structure, when $C_1 \subseteq C_2$ (a), when $C_2 \subseteq C_1$ (b), and when there is no subset relation between C_1 and C_2 (c).

respectively. If $C_1 \subseteq C_2$, the triplet $gc(\theta_1, \theta_2)$, if valid, will be inserted in an element in the interval $\mathcal{L}([C_1; C_2])$, as illustrated in Fig. 2(a). If, on the other hand, $C_2 \subseteq C_1$, the result $gc(\theta_1, \theta_2)$, if valid, will be inserted in $\mathcal{L}(C_1)$, as illustrated in Fig. 2(b). We note that in both cases the resulting triplet cannot be inserted lower in the lattice than at the level of the element $\mathcal{L}(C_1)$; it also cannot be inserted higher in the lattice than at the level of the largest of the two separating sets. From the proposition we further have that a valid triplet that results from applying the *gc*-operator to two triplets taken from lattice elements $\mathcal{L}(C_1), \mathcal{L}(C_2)$ with C_1 and C_2 not in a subset relation, may be inserted higher in the lattice than at the highest level of $\mathcal{L}(C_1)$ and $\mathcal{L}(C_2)$; the dynamics in this case are illustrated in Fig. 2(c). At which level exactly the new triplet will be inserted, depends on the number of variables that the first argument A_1 of the one triplet has in common with the separating set C_2 of the other triplet. The dynamics of triplet-set maintenance as occasioned by the *gc*-operator thus show that newly derived triplets cannot be inserted arbitrarily high or arbitrarily low in the organisational lattice structure. In fact, newly derived triplets can never be inserted lower in the lattice structure than at the level of the smallest conditioning set. The highest level at which a new triplet can be inserted is moreover indirectly constrained by the highest level at which non-empty lattice elements reside.

4. Computational advantages for fast-closure computation

The lattice structure for organising triplet sets as described in the previous section, brings computational advantages for various problems on independence relations. In this section, we focus on the advantages for fast-closure computation for semi-graphoid independence relations.

4.1. Selecting triplets for application of the GC-operator

At the core of the state-of-the-art algorithm for fast-closure computation for semi-graphoid independence relations, lies application of the *GC*-operator to all suitable pairs of triplets in the triplet set at hand. Given a triplet θ_1, selecting appropriately paired triplets θ_2 for this purpose involves checking essentially all other triplets in this set. When the set would be maintained as a simple list therefore, selecting all suitable pairs would take quadratic time in the number of triplets involved. Experimental results show unfortunately that, especially in intermediate iterations of the algorithm, this latter number can be several orders of magnitude larger than the size of the resultant fast-closure basis (see for example [1]). When the triplet set is organised in our lattice structure, the selection of all suitable pairs will take less running time in practice. To support this observation, the following proposition states a property by which the search for appropriately paired triplets for a given triplet of interest, can be focused on just a designated part of the lattice structure.

Proposition 7. *Let V be a set of variables as before and let $\theta_1 = \langle A_1, B_1 \mid C_1 \rangle \in V^{(3)}$ with $X_1 = A_1 \cup B_1 \cup C_1$. Then, $GC(\theta_1, \theta_2) = \varnothing$ for all triplets $\theta_2 = \langle A_2, B_2 \mid C_2 \rangle \in V^{(3)}$ for which at least one of the following properties holds:*

(*i*) $C_2 \setminus X_1 \neq \varnothing$;
(*ii*) $A_1 \cup B_1 \subseteq C_2$.

Proof. To prove that $GC(\theta_1, \theta_2) = \varnothing$ if property (*i*) holds, we observe that the condition $C_2 \subseteq X_1$, and hence $C_2 \setminus X_1 = \varnothing$, must be satisfied for $gc(\theta_1, \theta_2)$ to be a valid triplet. This same condition has to be met also for the other three applications of the *gc*-operator involved in *GC* to yield a valid result. If $C_2 \setminus X_1 \neq \varnothing$ therefore, the set $GC(\theta_1, \theta_2)$ is empty, as stated in the proposition.

We now prove that $GC(\theta_1, \theta_2) = \varnothing$ if property (*ii*) holds. From $A_1 \subseteq C_2$ and the fact that $A_2 \cap C_2 = \varnothing$ by definition, we find that $A_1 \cap A_2 = \varnothing$. The condition $A_1 \cap A_2 \neq \varnothing$ for $gc(\theta_1, \theta_2)$ to be a valid triplet thereby is not met, and $gc(\theta_1, \theta_2)$ is undefined. As similar observations hold for the other applications of the *gc*-operator involved in *GC*, we conclude that $GC(\theta_1, \theta_2) = \varnothing$, as stated in the proposition. □

From property (*i*) of Proposition 7 we now have that, given a triplet θ_1, the search for appropriately paired triplets for application of the *GC*-operator can be restricted to the triplets stored in the interval $\mathcal{L}([\varnothing; X_1])$ of the organisational lattice

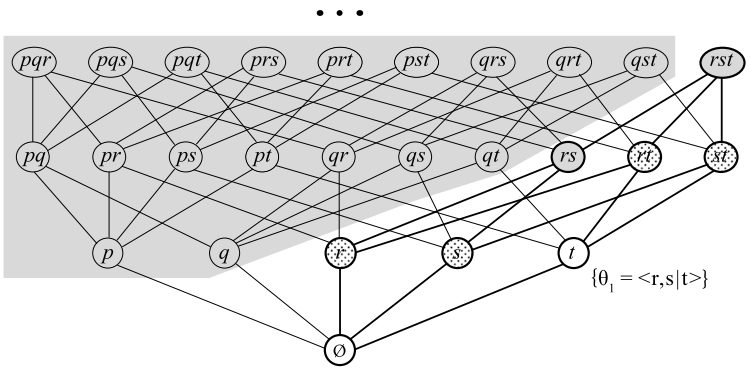

Fig. 3. A lattice structure over five variables, illustrating the computational implications of Proposition 7 and Corollary 8 for the selection of triplets for pairing with the triplet θ_1 for application of the *GC*-operator.

structure, as triplets stored at lattice elements outside this interval all have that $C_2 \setminus X_1 \neq \varnothing$. From the $2^{|V|} - (|V|+1)$ lattice elements of the (meet) lattice structure in total therefore, just $2^{|X_1|}$ elements remain for searching for appropriately paired triplets. The number of lattice elements to be searched is thereby reduced by a factor roughly equal to $2^{-|V \setminus X_1|}$. We note that, dependent of the size of the set X_1, this reduction can be quite substantial. By property (*ii*) of Proposition 7, we further have that the search for appropriately paired triplets for θ_1 can forego all triplets stored in the interval $\mathcal{L}([A_1 \cup B_1; V])$ of the lattice structure. We note that, as $A_1 \cup B_1 \subseteq X_1$, this second property may further restrict the interval $\mathcal{L}([\varnothing; X_1])$ of interest identified by the first property of the proposition.

Based on the proof of Proposition 7 we formulate the following corollary, which states that at specific elements of the interval of interest in the lattice, the search for appropriately paired triplets can be restricted to just a subset of the stored triplets.

Corollary 8. *Let V be as before and let $\theta_1 = \langle A_1, B_1 \mid C_1 \rangle \in V^{(3)}$. Then, for all triplets $\theta_2 = \langle A_2, B_2 \mid C_2 \rangle \in V^{(3)}$, the following properties hold:*

- *if $A_1 \subseteq C_2$, then $gc(\theta_1, \theta_2)$ and $gc(\theta_1, \theta_2^T)$ are undefined;*
- *if $B_1 \subseteq C_2$, then $gc(\theta_1^T, \theta_2)$ and $gc(\theta_1^T, \theta_2^T)$ are undefined.*

We illustrate the computational implications of Proposition 7 and its ensuing corollary on the search for appropriately paired triplets by means of a small example.

Example 4. We consider the variable set $V = \{p, q, r, s, t\}$ and the triplet $\theta_1 = \langle r, s \mid t \rangle$; the lattice structure associated with V is shown in Fig. 3. When searching for triplets θ_2 that can be appropriately paired with θ_1 for application of the *GC*-operator, we have from property (*i*) of Proposition 7 that the search can be restricted to the interval $\mathcal{L}([\varnothing; rst])$; the part of the lattice structure where such triplets θ_2 cannot reside by this property, is greyed out in the figure. From property (*ii*) of the proposition, we have that the lattice elements $\mathcal{L}(C_2)$ with $\{r, s\} \subseteq C_2$ also cannot contain any suitable triplets; these elements are individually indicated in dark grey in the figure. We note that the search for appropriately paired triplets θ_2 for θ_1 is thus restricted to just six lattice elements, out of the 26 elements of the lattice structure at large; the number of lattice elements to be inspected is thus reduced by a factor roughly equal to $2^{-|V \setminus X|} = \frac{1}{4}$. From Corollary 8, we further have that, for the remaining lattice elements $\mathcal{L}(C_2)$ with either $r \in C_2$ or $s \in C_2$, two of the four applications of the *gc*-operator involved in *GC* cannot yield a valid triplet; the lattice elements for which this property holds, are dotted in the figure. We conclude that, all in all, the search for appropriately paired triplets θ_2 for θ_1 is effectively restricted to two lattice elements in full and to half of the applications of the *gc*-operator for four lattice elements. □

While Proposition 7 addresses the pairing of triplets θ_2 with a given triplet θ_1, the following lemma identifies triplets θ_1 for which no θ_2 can yield a new, not yet g-included triplet. Upon fast-closure computation therefore, no search for paired triplets is required for these triplets θ_1.

Lemma 9. *Let V be as before and let $\theta_1 = \langle A_1, B_1 \mid C_1 \rangle \in V^{(3)}$ be a saturated triplet in which A_1, B_1 are singleton sets. Then, for all triplets $\theta_2 \in V^{(3)}$, each valid $\theta \in GC(\theta_1, \theta_2)$ has either $\theta = \theta_1$ or $\theta = \theta_1^T$.*

Proof. We prove the property stated in the lemma for $gc(\theta_1, \theta_2)$; the proofs for the other applications of the *gc*-operator involved in *GC* are analogous.

Let $\theta_2 \in V^{(3)}$ be such that $\theta = gc(\theta_1, \theta_2)$ is a valid triplet. By definition of the *gc*-operator, this triplet equals $\theta = \langle A, B \mid C \rangle = \langle A_1 \cap A_2, (B_2 \setminus C_1) \cup (B_1 \cap X_2) \mid C_1 \cup (A_1 \cap C_2) \rangle$. Since A_1 is a singleton set and $A = A_1 \cap A_2$ is not empty, we find

that $A = A_1$. By an analogous argument, we find that $A_1 \subseteq A_2$, which implies $A_1 \cap B_2 = \varnothing$ and $A_1 \cap C_2 = \varnothing$. We now observe that $B_2 \setminus C_1 \subseteq B_1$ since $C_1 = V \setminus (A_1 \cup B_1)$ and $A_1 \cap B_2 = \varnothing$, and that $B_1 \cap X_2 \subseteq B_1$. Since B is not empty, we thus find $B = B_1$. For the separating set C of θ, we find that $C = C_1 \cup (A_1 \cap C_2) = C_1$ since $A_1 \cap C_2 = \varnothing$. We conclude that $\theta = \langle A_1, B_1 \mid C_1 \rangle = \theta_1$. □

Lemma 9 pertains to saturated triplets θ_1 that reside at the highest level of the lattice structure at which triplets can be stored. It states that for these triplets there do not exist any triplets θ_2 that can give a valid, not yet g-included triplet upon application of the *GC*-operator. By this property therefore, all triplets stored at the $n \cdot (n-1)$ lattice elements at the highest level of the organisational structure, can be excluded as first arguments for the *GC*-operator upon fast-closure computation.

4.2. Checking for g-inclusion

The state-of-the-art algorithm for fast-closure computation for semi-graphoid independence relations in essence is an iterative procedure in which each iteration involves applying the *GC*-operator to all triplet pairs and subsequently removing all g-included triplets. Checking for g-inclusion is a computationally demanding step of the algorithm. When a triplet set would be maintained as a simple list to which newly generated triplets are appended, the overall check for g-inclusion would take quadratic time in the number of triplets stored, per iteration. When the triplet set is organised in our lattice structure, this check would in practice take less running time. For supporting this observation, we begin with formulating a property of transitivity of the g-inclusion relation among triplets.

Lemma 10. *Let V be as before and let $\theta_i = \langle A_i, B_i \mid C_i \rangle \in V^{(3)}$ with $X_i = A_i \cup B_i \cup C_i$, $i = 1, \ldots, 3$. If $\theta_1 \sqsubseteq \theta_2$ and $\theta_2 \sqsubseteq \theta_3$, then $\theta_1 \sqsubseteq \theta_3$.*

Proof. Without loss of generality, we assume throughout our proof that $\theta_i \sqsubseteq \theta_j$ implies $A_i \subseteq A_j$ and $B_i \subseteq B_j$. From the preconditions $\theta_1 \sqsubseteq \theta_2$ and $\theta_2 \sqsubseteq \theta_3$, we now find that

- $C_2 \subseteq C_1 \subseteq X_2$ and $C_3 \subseteq C_2 \subseteq X_3$; and
- $A_1 \subseteq A_2$, $B_1 \subseteq B_2$, $A_2 \subseteq A_3$ and $B_2 \subseteq B_3$.

From these properties, it readily follows that $A_1 \subseteq A_3$, $B_1 \subseteq B_3$ and $C_3 \subseteq C_1$. To show that $\theta_1 \sqsubseteq \theta_3$, we now only have to show that $C_1 \subseteq X_3$. From $A_1 \subseteq A_2$, $B_1 \subseteq B_2$ and $C_1 \subseteq X_2$, we find that $X_1 \subseteq X_2$; by a similar argument, we find that $X_2 \subseteq X_3$. It follows that $C_1 \subseteq X_1 \subseteq X_3$. By Definition 3, we conclude that the transitivity property as stated in the lemma, holds. □

The property of transitivity of the g-inclusion relation among triplets implies that if a newly generated triplet is g-included in a triplet of a non-redundant triplet set J, it cannot g-include any other triplets from this set. Reversely, if the new triplet g-includes a triplet from J, we know that it cannot be g-included itself by any other triplet from J. For a newly generated valid triplet θ_1, the following lemma now serves to characterise the separating sets of triplets that can possibly g-include θ_1 and the separating sets of triplets that can possibly be g-included by θ_1.

Proposition 11. *Let V be as before and let $\theta_i = \langle A_i, B_i \mid C_i \rangle \in V^{(3)}$, $i = 1, 2$. Then the following property holds for all $j, k = 1, 2$: if $\theta_j \sqsubseteq \theta_k$, then $C_k \subseteq C_j \subseteq X_k$, $A_k \setminus C_j \neq \varnothing$ and $B_k \setminus C_j \neq \varnothing$.*

Proof. We assume that $\theta_j \sqsubseteq \theta_k$, for some $j, k = 1, 2$. By Definition 3, we then have that $C_k \subseteq C_j \subseteq X_k$. From the definition it also follows that either $A_j \subseteq A_k$, $B_j \subseteq B_k$ or $B_j \subseteq A_k$, $A_j \subseteq B_k$. From $X_k = A_k \cup B_k \cup C_k$ and the observation that the sets A_j, B_j cannot be empty, we conclude that $A_k \setminus C_j \neq \varnothing$ and $B_k \setminus C_j \neq \varnothing$. □

When a triplet set is organised in our lattice structure as described in Section 3, we have from Proposition 11 that, for checking g-inclusion of a new triplet θ_1 with the separating set C_1, it needs to be compared only to triplets θ_2 stored in the interval $\mathcal{L}([\varnothing; C_1])$. Reversely, for checking which triplets are g-included by θ_1, the latter needs to be compared only to triplets θ_2 residing in the interval $\mathcal{L}([C_1; X_1])$; from this interval moreover, the lattice elements $\mathcal{L}(C_2)$ with $A_1 \subseteq C_2$ or $B_1 \subseteq C_2$, can be excluded from investigation.

Example 5. We consider again the variable set $\{p, q, r, s, t\}$ from Example 4, and its associated lattice structure in Fig. 4. We further consider a newly derived triplet $\theta_1 = \langle rs, t \mid p \rangle$ that is to be included in the intermediate non-redundant triplet set stored in the lattice structure. When searching for triplets θ_2 that g-include θ_1, we have by Proposition 11 that it suffices to consider the triplets in just the interval $\mathcal{L}([\varnothing; p])$. When searching for triplets θ_2 that are g-included by θ_1, we can focus on the interval $\mathcal{L}([p; prst])$. The part of the lattice that does not need to be searched for triplets with a g-inclusion relation with the new triplet θ_1 is greyed out in Fig. 4. We note that we also do not have to consider triplets in the elements $\mathcal{L}(C)$ of $\mathcal{L}([p; prst])$ for which we have that $\{r, s\} \subseteq \mathcal{L}(C)$ or $\{t\} \subseteq \mathcal{L}(C)$; these elements are individually indicated in grey in the

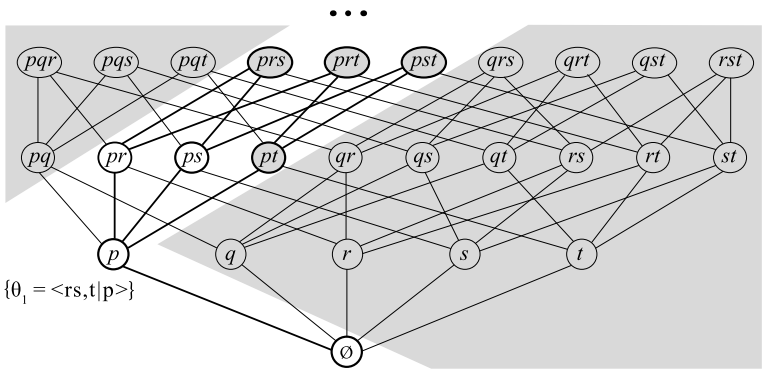

Fig. 4. A lattice structure over five variables, illustrating the computational implications of Proposition 11 for the selection of triplets for checking their g-inclusion relation with a new triplet θ_1.

figure. All in all, we find that the search in the lattice for triplets that have a g-inclusion relation with the new triplet θ_1 can be restricted to just four lattice elements, out of the 26 elements in total. □

5. Experiments and results

The state-of-the-art algorithm for fast-closure computation for semi-graphoid independence relations has a high runtime complexity in general, as discussed in Section 2. The two basic steps of the algorithm's main loop each take polynomial time in the number of triplets in the intermediate set per iteration, but as these intermediate sets can grow to a sizeable part of the full closure of the starting set, the overall running time may in fact be exponential in the starting number of triplets. In the previous sections, we introduced a lattice structure for organising triplet sets and argued that the properties of this organisational structure serve to reduce the running time of fast-closure computation in practice. The insights gained from the new triplet-set organisation however, do not affect the algorithm's worst-case running time. To investigate the computational advantages of using our lattice structure on the running times in practical settings, we conducted a range of experiments.

5.1. Experimental set-up

We briefly describe the set-up of our experiments before presenting our results.

The implementations In our experiments, we compared the performances of essentially two implementations[1] of the state-of-the-art fast-closure algorithm for semi-graphoid independence relations in general, as outlined in Algorithm 1. The *FC implementation*, or *FC* for short, is a previously existing implementation of the fast-closure algorithm, that employs a straightforward representation of triplet sets; this implementation is described in further detail in [2]. The *LFC implementation*, or *LFC* for short, is a new implementation which exploits our lattice structure for organising triplet sets, as described in the previous sections. While both implementations follow the outline of the basic algorithm of Algorithm 1, LFC explicitly incorporates the results from the Sections 3.2 and 4. More specifically, it exploits Proposition 7, Corollary 8 and Lemma 9 to reduce the computations involved in line 6 of the basic algorithm; we recall that these properties serve to avoid, as much as possible, applications of the *gc*-operator that cannot yield any new, valid triplets. The LFC implementation further builds upon Proposition 6 with respect to the lattice element at which a newly generated triplet is to be entered, and exploits, in line 7 of the basic algorithm, Proposition 11 for focusing identification of triplets that are in a g-inclusion relation with the newly entered triplet.

The FC implementation is a heavily optimised implementation used in various experimental settings, and the new LFC implementation developed for the current paper may not incorporate the same advanced level of optimisation. This difference may influence the running times of the two implementations in a way that is unrelated to the exploitation of our organisational lattice structure. To allow for a proper comparison of implementations with and without exploiting the properties of the lattice structure, we constructed also a stripped version of LFC, called the *LFC$^-$ implementation*, that uses the lattice structure for storing triplets, yet does not exploit any of its associated properties. Upon verifying whether a newly generated triplet is g-included in an already available triplet for example, LFC$^-$ will examine all available triplets, without using Proposition 11 to restrict the number of triplets for investigation.

[1] The two implementations can be downloaded from https://github.com/mbaioletti/fast_closure and https://github.com/mbaioletti/lattice_fast_closure, respectively.

The instance generator We implemented an *instance generator* for generating starting sets of triplets, called *instances*, on which the FC, LFC and LFC$^-$ implementations of the fast-closure algorithm will be run. The instance generator takes four parameters, which are the number of variables n involved, the number of triplets t in the starting set, and two probabilities p and q with $2p + q < 1$. The role of the two probability parameters is to govern the composition of the triplets in the generated starting sets. More specifically, in a generated triplet $\theta = \langle A, B \mid C \rangle$, each variable V_i occurs in the set A with probability p, in B with probability p, and in the triplet's separating set C with probability q; with probability $1 - 2p - q$, therefore, the variable V_i does not occur in the triplet θ. We used the two probability parameters to compare the performances of our implementations on both simpler and harder instances.

The experiments were conducted with $n = 10, 20, 30, 40, 50, 60, 70, 80$ variables and starting sets of $t = \lfloor 1.5n \rfloor, 2n, 3n, 5n, 10n$ triplets; for each value of n and associated t value, we generated 100 instances. We further chose two different value combinations for the parameters p and q. As the first combination, we set $p = 0.2$ and $q = 0.3$; this parameter setting results in triplets in which the separating set C on average is larger in size than the sets A and B. For the second combination, we set $p = q = 0.25$, resulting in triplets in which the size of the separating set C is comparable to that of the sets A and B. In order to complete the experiments in a reasonable amount of time, we set a time limit of 1200 seconds per instance: if, for a generated instance, no fast-closure basis was returned within this time limit, the computation was halted and the instance discounted.

The experiments Our experiments were organised as two sets of runs. The first set of runs were aimed at gaining insight in the extent to which our organisational lattice structure and its associated properties help reduce the overall burden of fast-closure computation. In this set of runs, we compared the running times of the FC implementation to those of LFC. The results from these runs are summarised in Tables 1 and 2, for the two settings of the probability parameters p, q, respectively. The tables report, for each implementation, the number of instances S_i for which it was able to establish a fast-closure basis within the imposed time limit; also, the average running time $\bar{T}_i$ is reported, computed for each implementation over the instances it *solved* within the allotted time. The column r_T, to conclude, reports the percentage ratio of the running time of the LFC implementation against the running time of FC, computed over the instances that were solved within the time limit by *both* implementations.

In the second set of runs, we compared the LFC implementation to its stripped LFC$^-$ version, with respect to the number of g-inclusion checks performed and the number of applications of the *gc*-operator. The results from this second set of runs are summarised in Tables 3 and 4, for the two settings of the probability parameters p, q, respectively. The tables report, for each implementation, the number of instances S_i for which it was able to establish a fast-closure basis within the allotted limit. The tables further give, for each implementation, the average running time T_i, the average number of g-inclusion checks I_i performed, and the average number of applications of the *gc*-operator O_i, with these statistics computed over the instances that were *solved* in time. The column r_T has the same meaning as that in the Tables 1 and 2. The column r_I further reports the percentage ratio of the average number of g-inclusion checks performed by the LFC implementation against that performed by LFC$^-$, computed over just the instances that were solved within the time limit by *both* implementations, and the column r_O reports a similar percentage ratio for the number of applications of the *gc*-operator.

5.2. Results

We present and discuss the results obtained from the two sets of runs separately.

First set of runs We recall that the first set of experimental runs were performed to gain insight in the extent to which our organisational structure for triplet sets helps reduce the overall burden of fast-closure computation. The results in Table 1, obtained from instances with $p = 0.2, q = 0.3$, show that the LFC implementation was faster than FC and was able to solve a larger number of instances, for each parameter combination n, t.

With respect to running times, we observe that, in general, for each value of n, the averaged running time of fast-closure computation increases with the number of starting triplets t. This finding conforms to expectation since, roughly spoken, more starting triplets tend to give more triplet combinations to which the *gc*-operator can be applied, which in turn may yield more new triplets for the algorithm's next iteration. The only evident exception was found for instances with the parameter combination $n = 10, t = 100$, on which both implementations were considerably faster than on instances with $n = 10, t = 50$. A possible explanation for this finding is that, for a fixed n, increasing the number of starting triplets beyond a specific t value can make instances become simpler as the number of new triplets per iteration may tend to decrease and as a consequence the number of iterations may decrease as well; the inability of both FC and LFC to solve all generated instances within the time limit however, shows that still some very hard cases can result from our generating scheme. Further experimental studies, with increasing numbers of starting triplets per n value, are required however, to support the existence of a (possibly steep) phase transition in hardness of instances.

The number of instances solved within the time limit by the LFC implementation was 3 638, while FC solved 3 594 instances from among the total of 4 000 instances offered as input to the two implementations. For the n, t value combinations with $n = 20, 100 \leq t \leq 200$, neither of the implementations were able to complete the fast-closure computation within the alloted time for any of the generated instances. Further experiments, not detailed here, revealed that establishing the fast-closure bases for these instances required more than 3 600 seconds each, thereby showing that these instances were

Table 1
Experimental results for the average running times and the numbers of successfully completed instances, obtained with the LFC and FC implementations respectively, on instances with $p = 0.2, q = 0.3$.

Instance		*LFC*		*FC*		*Summary* (in %)
n	t	S_1	$\bar{T}_1$	S_2	$\bar{T}_2$	r_T
10	15	100	0.29224	100	1.70410	17.15
10	20	100	1.93784	100	16.21171	11.95
10	30	100	28.34389	94	148.78038	13.18
10	50	100	123.42190	88	203.44953	40.30
10	100	99	42.11258	95	89.16604	44.80
20	30	99	0.14763	99	1.04615	14.11
20	40	95	42.17774	89	16.03187	10.84
20	60	48	52.26565	43	110.10540	9.49
30	45	100	0.05494	100	0.35082	15.66
30	60	100	0.09120	100	0.57073	15.98
30	90	100	3.98559	99	0.91641	14.00
30	150	88	42.66994	82	51.81888	10.79
30	300	10	111.01060	6	663.46950	5.39
40	60	100	0.00066	100	0.00401	16.46
40	80	100	0.00182	100	0.01328	13.70
40	120	100	0.01012	100	0.05809	17.42
40	200	100	0.02813	100	0.24539	11.46
40	400	99	3.29188	99	26.89525	12.24
50	75	100	0.00018	100	0.00268	6.72
50	100	100	0.00038	100	0.00629	6.04
50	150	100	0.00148	100	0.01655	8.94
50	250	100	0.00872	100	0.07938	10.99
50	500	100	0.05348	100	0.43107	12.41
60	90	100	0.00009	100	0.00316	2.85
60	120	100	0.00004	100	0.00550	0.73
60	180	100	0.00060	100	0.01519	3.95
60	300	100	0.00297	100	0.04626	6.42
60	600	100	0.02202	100	0.21501	10.24
70	105	100	0.00000	100	0.00409	0.00
70	140	100	0.00003	100	0.00780	0.38
70	210	100	0.00059	100	0.01860	3.17
70	350	100	0.00295	100	0.05599	5.27
70	700	100	0.01329	100	0.22023	6.03
80	120	100	0.00003	100	0.00550	0.55
80	160	100	0.00000	100	0.00979	0.00
80	240	100	0.00092	100	0.02542	3.62
80	400	100	0.00367	100	0.07441	4.93
80	800	100	0.01290	100	0.26756	4.82

particularly hard to solve. Also the instances with $n = 30, t = 300$ proved difficult to solve: the LFC implementation solved just ten of these instances within the time limit, while FC successfully completed only six of them. With increasing n values and the linearly related numbers of starting triplets t used in the experiments, the instances become easier and easier to solve, with the average times decreasing to less than one second. The most plausible explanation of this latter finding is that, for larger values of n, with the associated t values used, the likelihood of the *gc*-operator being applicable to two triplets gets smaller and smaller. This explanation is corroborated by our finding that the established fast-closure basis for such larger instances is often equal to the starting set of triplets or includes just a quite small percentage of new triplets. Further experimental investigation of the t values in relation to the value of n, is required to establish at which values of t as a function of n, non-trivial fast-closure bases begin to result.

For the percentage ratio r_T of the running time of the LFC implementation against that of FC, we note that a higher reported value indicates that the gain in efficiency of LFC over FC is smaller; a ratio larger than 100% would in fact indicate that FC would be more efficient than LFC over the solved instances. The highest value of r_T was found to be 44.8% however, for the instances with $n = 10, t = 100$. The small and even decreasing values for the ratio r_T with increasing values of n, demonstrate that the LFC implementation is much faster than FC, at least so for the easier instances. As the ratios r_T reported in Tables 1 and 2 include efficiency effects on the differences in running times that are unrelated to the exploitation of our organisational lattice structure, we will return to this observation in our discussion of the results of the second set of experimental runs.

Table 2
Experimental results for the average running times and the numbers of successfully completed instances, obtained with the LFC and FC implementations respectively, on instances with $p = 0.25, q = 0.25$.

Instance		*LFC*		*FC*		*Summary* (in %)
n	t	S_1	$\bar{T}_1$	S_2	$\bar{T}_2$	r_T
10	15	100	3.39615	100	17.10122	19.86
10	20	100	15.42097	98	63.78513	21.24
10	30	100	61.40180	97	186.28246	29.91
10	50	100	53.78122	95	110.73517	30.38
10	100	100	16.52861	100	24.01262	68.83
20	30	16	95.22219	10	31.05820	6.46
30	45	64	83.56870	50	154.03338	5.92
30	60	23	102.37674	14	154.41871	7.05
40	60	96	12.53974	94	21.15840	7.38
40	80	84	50.26715	79	65.36171	7.80
40	120	47	154.03609	32	284.50275	6.13
50	75	100	0.06766	100	0.76475	8.85
50	100	100	1.44286	100	15.96336	9.04
50	150	98	12.10201	94	46.52380	6.87
50	250	68	65.37225	56	185.39348	5.65
60	90	100	0.01002	100	0.07698	13.02
60	120	100	0.01664	100	0.14922	11.15
60	180	99	0.15801	99	1.49345	10.58
60	300	98	2.36136	98	25.20788	9.37
60	600	82	103.51238	57	233.96428	6.42
70	105	100	0.00192	100	0.02184	8.79
70	140	100	0.00420	100	0.04426	9.49
70	210	100	0.00955	100	0.11276	8.47
70	350	100	0.04834	100	0.63682	7.59
70	700	100	2.19582	100	21.85140	10.05
80	120	100	0.00059	100	0.01675	3.52
80	160	100	0.00135	100	0.02601	5.19
80	240	100	0.00688	100	0.08797	7.82
80	400	100	0.01517	100	0.21788	6.96
80	800	100	0.12882	100	1.44934	8.89

The results in Table 2, obtained from instances with the parameter setting $p = 0.25, q = 0.25$, are in line with those reported in Table 1, yet reveal that the generated instances are more difficult to solve in general than the instances generated with $p = 0.2, q = 0.3$. The averaged running times of the LFC and FC implementations reported in Table 2 for instances with $n \geq 20$, are consistently higher than those reported in Table 1. For the 47 instances with $n = 40, t = 120$, solved within the time limit, for example, Table 2 reports that LFC used some 154 seconds on average, while Table 1 reveals that all instances with this same combination of n, t values, yet generated with the parameter setting $p = 0.2, q = 0.3$, were particularly easy, requiring less than one second for their solution on average. Of the total of 4 000 instances offered to the two implementations, LFC and FC successfully completed fast-closure computation for 2 675 and 573 instances, respectively, within the allotted time. Of the 100 instances with $n = 20, t = 30$, the LFC implementation could solve only sixteen instances in time and FC just ten. Both implementations were not able to complete the fast-closure computation for any of the instances with $n = 20, 40 \leq t \leq 200$, and also were not able to solve any instance with $n = 30, 90 \leq t \leq 300$, with $n = 40, 200 \leq t \leq 400$ or with $n = 50, t = 500$.

Second set of runs We recall that the second set of experimental runs were aimed at gaining insight in the computational gains from the properties of the organisational lattice structure, as discussed in the Sections 3.2 and 4. For this purpose, the LFC implementation and its stripped LFC$^-$ version were compared as to the number of g-inclusion checks performed and the number of applications of the *gc*-operator. The experimental setting for this second set of runs was the same as for the first set of runs: the same values of the parameters n, t, p, q, the same number of instances and the same time limit were used. The results in Table 3, obtained from instances with $p = 0.2, q = 0.3$, demonstrate the computational advantages of exploiting the various properties of the organisational structure. With respect to running times, we observe that LFC is consistently faster on the instances that were solved by both implementations than LFC$^-$. We also observe that LFC performs fewer g-inclusion checks and applications of the *gc*-operator than LFC$^-$. Especially for the larger instances, the difference between LFC and LFC$^-$ in terms of these basic operations is quite striking, with LFC often performing fewer than 10% of the operations performed by LFC$^-$. The results in Table 4, obtained from instances with the parameter setting

Table 3
Experimental results for the number of g-inclusion checks and *gc*-applications, obtained with the LFC and LFC$^-$ implementations respectively, on instances with $p = 0.2, q = 0.3$.

Instance		*LFC*				*LFC*$^-$				*Summary* (in %)		
n	t	S_1	$\bar{T}_1$	$\bar{I}_1$	$\bar{O}_1$	S_2	$\bar{T}_2$	$\bar{I}_2$	$\bar{O}_2$	r_T	r_I	r_O
10	15	100	5.51300	5.480e+07	1.382e+06	100	9.00174	1.806e+08	1.879e+06	61.24	30.34	73.53
10	20	100	35.71958	3.838e+08	5.091e+06	100	55.64773	1.200e+09	6.622e+06	64.19	31.98	76.88
10	30	98	99.87797	1.027e+09	1.112e+07	97	128.72476	2.631e+09	1.227e+07	67.88	35.02	88.62
10	50	100	102.72593	1.092e+09	1.417e+07	100	146.14107	3.177e+09	1.547e+07	70.29	34.39	91.60
10	100	100	43.34913	4.950e+08	7.802e+06	100	47.51959	8.751e+08	8.040e+06	91.22	56.57	97.04
20	30	13	258.20675	2.736e+09	5.447e+07	10	134.62556	3.511e+09	6.488e+07	27.47	7.12	23.40
30	45	57	79.87952	6.276e+08	2.066e+07	51	136.92963	3.390e+09	4.886e+07	24.45	6.68	20.65
30	60	16	121.69160	9.354e+08	2.972e+07	11	62.01840	1.582e+09	3.303e+07	20.56	5.06	16.39
40	60	100	9.09404	7.579e+07	1.969e+06	98	2.84747	7.153e+07	1.996e+06	31.27	8.59	20.87
40	80	84	39.91208	2.844e+08	7.559e+06	81	60.17316	1.562e+09	1.913e+07	31.63	8.74	22.08
40	120	35	144.29563	9.504e+08	2.845e+07	30	277.70144	6.527e+09	1.095e+08	24.58	6.38	14.67
50	75	100	5.44947	8.054e+07	5.393e+05	98	0.09967	1.544e+06	1.635e+05	36.20	8.44	17.06
50	100	100	0.30131	1.626e+06	1.371e+05	100	0.70042	1.544e+07	6.603e+05	43.02	10.53	20.77
50	150	98	12.14560	9.848e+07	2.175e+06	97	24.23507	5.809e+08	9.720e+06	26.35	7.12	15.11
50	250	63	133.68460	7.769e+08	1.825e+07	53	217.00852	5.455e+09	8.208e+07	24.26	5.18	10.67
60	90	100	0.00539	8.044e+03	3.836e+03	100	0.01567	1.042e+05	5.643e+04	34.40	7.72	6.80
60	120	100	0.01332	1.391e+04	6.856e+03	100	0.03502	2.294e+05	1.128e+05	38.04	6.06	6.08
60	180	100	0.40491	1.575e+06	9.534e+04	100	1.15226	2.658e+07	7.784e+05	35.14	5.92	12.25
60	300	100	2.86227	1.323e+07	5.512e+05	100	8.54102	2.205e+08	4.236e+06	33.51	6.00	13.01
60	600	74	184.95981	6.901e+08	1.672e+07	58	161.86385	4.104e+09	8.091e+07	24.69	2.42	5.87
70	105	100	0.00160	1.020e+03	1.141e+03	100	0.00960	2.904e+04	5.153e+04	16.67	3.51	2.21
70	140	100	0.00428	2.727e+03	2.396e+03	100	0.02337	7.539e+04	9.863e+04	18.31	3.62	2.43
70	210	100	0.01030	5.502e+03	5.344e+03	100	0.06198	2.163e+05	2.428e+05	16.62	2.54	2.20
70	350	100	0.15701	4.101e+05	3.604e+04	100	0.43872	6.464e+06	8.845e+05	35.79	6.34	4.07
70	700	100	6.21826	2.563e+07	7.749e+05	100	15.16104	4.095e+08	9.090e+06	41.01	6.26	8.53
80	120	100	0.00060	2.971e+02	5.246e+02	100	0.00799	1.984e+04	5.960e+04	7.51	1.50	0.88
80	160	100	0.00210	9.457e+02	1.209e+03	100	0.01847	4.952e+04	1.113e+05	11.37	1.91	1.09
80	240	100	0.00599	1.628e+03	2.438e+03	100	0.04586	1.193e+05	2.563e+05	13.06	1.36	0.95
80	400	100	0.03464	5.248e+04	1.087e+04	100	0.21906	1.254e+06	7.736e+05	15.82	4.18	1.41
80	800	100	0.18260	1.356e+05	3.902e+04	100	1.12584	6.950e+06	3.604e+06	16.22	1.95	1.08

$p = 0.25, q = 0.25$, are much in line with those reported in Table 3, and in fact demonstrate that the computational gains derived from properties of the organisational lattice structure are larger for the harder instances.

To conclude, we take a closer look at the instances with $n = 10, t \geq 50$. Both Tables 3 and 4 show that the numbers of applications of the *gc*-operator by the LFC and LFC$^-$ implementations are more or less the same, with LFC$^-$ applying the operator just a little more often than LFC. A possible explanation for this finding is that, with this combination of n, t values and the parameter values for p, q under study, many instances are generated for which the fast-closure basis is composed of saturated triplets with an empty separating set. With the parameter setting $n = 10, t = 100, p = 0.2, q = 0.3$, for example, this phenomenon was seen in 89 of the 100 generated instances. For such instances, the triplets that are generated upon fast-closure computation, tend to get concentrated in a single element of the lattice structure, as a result of which the organisation of the basis under construction is similar to a simple list and the properties of the lattice structure do not bring any computational advantages. To corroborate this observation, Table 5 describes the dynamics of the organisational lattice structure for a single run of the LFC implementation on an instance with the parameter setting under consideration. After each iteration of the basic algorithm for fast-closure computation, the table reports the number of non-empty elements in the lattice and the height of the lattice, that is, the highest level at which a non-empty element resides, plus one. The table shows that, in the run under study, triplets initially are distributed over 148 elements on the first seven levels of the lattice structure. As more iterations are executed, the number of non-empty lattice elements, and hence the number of different separating sets occurring in the triplets, are reduced to just eight after the fifth iteration and eventually fall to one. Simultaneously, the height of the lattice structure is reduced to one, indicating that the fast-closure basis computed for the instance under study is composed of triplets with the empty set for their separating set. In this run, the number of triplets in the fast-closure basis equals the 511 which is the maximum number of saturated triplets with an empty separating set.

6. Concluding observations

Despite important advances in the past decades, fast-closure computation for (semi-graphoid) independence relations is still computationally challenging in general. In this paper, we have proposed a lattice-based representation for triplet sets

Table 4
Experimental results for the number of g-inclusion checks and *gc*-applications, obtained with the LFC and LFC$^-$ implementations respectively, on instances with $p = 0.25, q = 0.25$.

Instance		*LFC*				*LFC*$^-$				*Summary* (in %)		
n	t	S_1	$\bar{T}_1$	$\bar{I}_1$	$\bar{O}_1$	S_2	$\bar{T}_2$	$\bar{I}_2$	$\bar{O}_2$	r_T	r_I	r_O
10	15	100	0.25641	2.712e+06	1.475e+05	100	0.42638	1.116e+07	2.395e+05	60.14	24.31	61.58
10	20	100	3.18333	3.242e+07	1.215e+06	100	5.70802	1.241e+08	2.039e+06	55.77	26.12	59.57
10	30	100	37.32756	3.873e+08	7.884e+06	100	62.01636	1.553e+09	1.061e+07	60.19	24.94	74.33
10	50	97	206.72493	2.139e+09	2.560e+07	95	277.71476	5.334e+09	2.562e+07	69.05	36.60	91.23
10	100	95	134.78078	1.356e+09	2.399e+07	93	208.42896	4.630e+09	2.483e+07	61.21	26.94	96.64
20	30	100	2.56051	2.377e+07	1.040e+06	100	7.61312	2.288e+08	3.190e+06	33.63	10.39	32.61
20	40	92	9.96577	9.538e+07	3.256e+06	91	13.32583	3.365e+08	6.352e+06	40.09	16.61	29.01
20	60	56	53.96527	4.506e+08	1.154e+07	53	42.41975	8.836e+08	2.350e+07	28.82	8.75	21.96
20	100	1	0.42700	4.558e+05	4.001e+05	1	2.37900	1.214e+07	3.373e+06	17.95	3.76	11.86
30	45	100	0.00661	1.345e+04	5.950e+03	100	0.01393	1.032e+05	3.470e+04	47.45	13.03	17.15
30	60	100	0.46784	3.359e+06	2.240e+05	100	1.44357	3.932e+07	7.698e+05	32.41	8.54	29.10
30	90	100	1.60082	1.012e+07	5.845e+05	100	5.21942	1.357e+08	2.173e+06	30.67	7.46	26.90
30	150	96	12.82293	1.001e+08	3.461e+06	96	50.72901	1.323e+09	1.868e+07	25.28	7.57	18.52
30	300	11	126.18510	5.906e+08	3.220e+07	8	315.79800	5.326e+09	2.122e+08	14.86	2.63	7.04
40	60	100	0.00083	9.372e+02	9.954e+02	100	0.00358	1.346e+04	1.894e+04	23.29	6.96	5.26
40	80	100	0.00184	1.945e+03	1.926e+03	100	0.00836	3.354e+04	3.755e+04	22.07	5.80	5.13
40	120	100	0.00961	3.760e+04	7.273e+03	100	0.03327	3.571e+05	1.018e+05	28.89	10.53	7.15
40	200	100	0.04449	6.820e+04	2.406e+04	100	0.15599	1.603e+06	4.253e+05	28.52	4.25	5.66
40	400	97	8.72945	3.526e+07	1.648e+06	97	25.34841	6.256e+08	1.182e+07	34.44	5.64	13.95
50	75	100	0.00010	8.854e+01	2.591e+02	100	0.00253	6.744e+03	2.276e+04	3.95	1.31	1.14
50	100	100	0.00051	3.034e+02	5.713e+02	100	0.00589	1.543e+04	4.222e+04	8.68	1.97	1.35
50	150	100	0.00182	7.886e+02	1.424e+03	100	0.01770	4.302e+04	9.943e+04	10.30	1.83	1.43
50	250	100	0.00848	2.252e+03	4.128e+03	100	0.06682	1.583e+05	2.963e+05	12.69	1.42	1.39
50	500	100	0.07850	6.525e+04	2.522e+04	100	0.46577	3.128e+06	1.470e+06	16.85	2.09	1.72
60	90	100	0.00007	1.159e+01	1.255e+02	100	0.00444	8.255e+03	3.216e+04	1.50	0.14	0.39
60	120	100	0.00030	1.692e+02	3.189e+02	100	0.00723	1.751e+04	5.861e+04	4.15	0.97	0.54
60	180	100	0.00108	2.433e+02	6.245e+02	100	0.01699	3.917e+04	1.321e+05	6.34	0.62	0.47
60	300	100	0.00471	5.498e+02	1.623e+03	100	0.05429	1.161e+05	3.709e+05	8.68	0.47	0.44
60	600	100	0.02797	2.047e+03	6.628e+03	100	0.33687	5.562e+05	1.530e+06	8.30	0.37	0.43
70	105	100	0.00009	0.000e+00	6.258e+01	100	0.00460	1.092e+04	4.368e+04	1.93	0.00	0.14
70	140	100	0.00018	2.359e+01	1.243e+02	100	0.00862	1.998e+04	7.806e+04	2.06	0.12	0.16
70	210	100	0.00162	7.487e+01	2.910e+02	100	0.02026	4.619e+04	1.767e+05	8.01	0.16	0.16
70	350	100	0.00334	7.077e+01	7.116e+02	100	0.05472	1.255e+05	4.902e+05	6.11	0.06	0.15
70	700	100	0.01750	3.649e+02	2.886e+03	100	0.25651	5.295e+05	1.975e+06	6.82	0.07	0.15
80	120	100	0.00010	0.000e+00	2.938e+01	100	0.00648	1.428e+04	5.712e+04	1.54	0.00	0.05
80	160	100	0.00030	0.000e+00	5.653e+01	100	0.01071	2.544e+04	1.018e+05	2.80	0.00	0.06
80	240	100	0.00127	0.000e+00	1.179e+02	100	0.02412	5.736e+04	2.294e+05	5.25	0.00	0.05
80	400	100	0.00350	2.359e+01	3.476e+02	100	0.06911	1.610e+05	6.390e+05	5.06	0.01	0.05
80	800	100	0.01586	9.436e+01	1.419e+03	100	0.29046	6.503e+05	2.562e+06	5.46	0.01	0.06

Table 5
The dynamics of the lattice structure, per iteration of the basic algorithm, for a run of the LFC implementation on an instance with $n = 10, t = 100, p = 0.2, q = 0.3$.

iteration	*# elements*	*height*
1	148	7
2	181	6
3	147	6
4	70	5
5	8	3
6	1	1
7	1	1

to support more efficient computation of fast-closure bases in practice. This lattice representation allowed us to study the structure of triplet sets in general. From the insights gained, we formulated various useful properties, for example for reducing the computations involved in maintaining a non-redundant triplet set by allowing redundancy checks to be restricted to parts of the lattice and for effectively selecting triplets for application of the main operator involved in fast-closure computation. In an experimental setting, we compared different implementations of the state-of-the-art algorithm for fast-

closure computation, with and without using the supporting lattice structure; for the comparison of these implementations, an instance generator was used, allowing properties of the generated instances of triplet sets to be governed by multiple parameters. The results obtained from our experiments demonstrate that the computational advantages of using the new lattice-based representation for independence relations can be substantial in practice.

Having studied the advantages of our lattice representation for semi-graphoid independence relations, we are now extending our investigations to other types of relation, among which are the graphoid independence relations; we expect that triplet sets of different types of relation embed more structural properties than triplet sets of semi-graphoid relations in general, which in turn may be exploited to further reduce computation times. In addition, we plan to further investigate such structural properties from a more foundational perspective, with the ultimate goal of arriving at algorithms for manipulating independence relations with improved properties of scalability. For the same goal, we further intend to expand our experimental studies, by investigating the hardness of instances of triplet sets, in terms of the parameters governing the composition of the triplets themselves and of the sizes of the starting sets, for fast-closure computation.

Declaration of competing interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

Acknowledgements

The research reported in this paper was conducted during a sabbatical stay of the first author at Università di Perugia, Italy. The current paper is a revised and extended version of the conference paper presented at the 9th International Conference on Probabilistic Graphical Models (PGM 2018), held in Prague, Czech Republic: L.C. van der Gaag, M. Baioletti, J.H. Bolt. A lattice representation of independence relations. *Proceedings of Machine Learning Research: International Conference on Probabilistic Graphical Models*, vol. 72, pp. 487–498. We would like to express our gratitude to all reviewers who have provided us with their highly valuable feedback.

References

[1] M. Baioletti, G. Busanello, B. Vantaggi, Closure of independencies under graphoid properties: some experimental results, in: International Symposium on Imprecise Probability: Theories and Applications, Durham, 2009, pp. 11–19.
[2] M. Baioletti, G. Busanello, B. Vantaggi, Conditional independence structure and its closure: inferential rules and algorithms, Int. J. Approx. Reason. 50 (2009) 1097–1114.
[3] G. Birkhoff, Lattice Theory, 3rd edition, AMS Colloquium Publications, vol. 25, 1995.
[4] R.G. Cowell, A.P. Dawid, S.L. Lauritzen, D.J. Spiegelhalter, Probabilistic Networks and Expert Systems: Exact Computational Methods for Bayesian Networks, Springer, New York, 2003.
[5] A.P. Dawid, Conditional independence in statistical theory, J. R. Stat. Soc. B 41 (1979) 1–31.
[6] D. Koller, N. Friedman, Probabilistic Graphical Models: Principles and Techniques, MIT Press, Cambridge, 2009.
[7] S. Lopatatzidis, L.C. van der Gaag, Concise representations and construction algorithms for semi-graphoid independency models, Int. J. Approx. Reason. 80 (2017) 377–392.
[8] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann, 1988.
[9] M. Studený, Conditional independence relations have no finite complete characterization, in: Information Theory, Statistical Decision Functions and Random Processes, Kluwer, Dordrecht, 1992, pp. 377–396.
[10] M. Studený, Semigraphoids and structures of probabilistic conditional independence, Ann. Math. Artif. Intell. 21 (1997) 71–98.
[11] M. Studený, Complexity of structural models, in: Proceedings of the Joint Session of the 6th Prague Conference on Asymptotic Statistics and the 13th Prague Conference on Information Theory, Statistical Decision Functions and Random Processes, Prague, 1998, pp. 521–528.
[12] M. Studený, Probabilistic Conditional Independence Structures, Springer Verlag, London, 2005.