CEUR-WS.org/Vol-2756/paper_20.pdf

Hierarchical Decompositions of Dihypergraphs*

Lhouari Nourine and Simon Vilmin

LIMOS, Université Clermont Auvergne, Aubière, France
simon.vilmin@ext.uca.fr, lhouari.nourine@uca.fr

Abstract. In this paper we are interested in decomposing a dihypergraph $\mathcal{H} = (\mathrm{V}, \mathcal{E})$ into simpler dihypergraphs, that can be handled more efficiently. We study the properties of dihypergraphs that can be hierarchically decomposed into trivial dihypergraphs, i.e., vertex hypergraph. The hierarchical decomposition is represented by a full labelled binary tree called $\mathcal{H}$-tree, in the fashion of hierarchical clustering. We present a polynomial time and space algorithm to achieve such a decomposition by producing its corresponding $\mathcal{H}$-tree. However, there are dihypergraphs that cannot be completely decomposed into trivial components. Therefore, we relax this requirement to more indecomposable dihypergraphs called H-factors, and discuss applications of this decomposition to closure systems and lattices.

Keywords: Dihypergraphs · Decomposition · Closure systems · Lattices

1 Introduction

In this paper we are interested in decomposing *directed hypergraphs* (*dihypergraphs* for short). They are a generalization of directed graphs, as hypergraphs generalize graphs. Dihypergraphs are often used to model implication systems in various fields of computer science such as databases [ADS86, AL17], closure systems and lattice theory [BDVG18, Wil17], propositional and Horn logic [GLPN93, GGPR98, Wil17] for instance.

A dihypergraph consists in a finite set of vertices V and a collection $\mathcal{E}$ of (hyper)arcs (sometimes called hyperedges) of the form (B, h) over V, where B is a subset of V and h a singleton of V. In database theory, V corresponds to a relation schema and arcs are functional dependencies; whereas in Horn logic an arc is definite Horn clause on the propositional variables set V. In general, an arc (B, h) depicts a causality relation between B and h, namely, whenever we deal with B we also have to take h into consideration. Note however that a more general definition of dihypergraph is given in [GLPN93, GGPR98] where the dihypergraphs we use in this paper are called B-graphs.

We are interested in decomposing a dihypergraph $\mathcal{H} = (\mathrm{V}, \mathcal{E})$ into simpler dihypergraphs, that can be handled more efficiently. The *hierarchical decomposition* (H-decomposition for short) of a dihypergraph considered in this paper, is

* Copyright © 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0). The second author is funded by the CNRS, France, ProFan project.

a *recursive partitioning* of the vertex set of the dihypergraph into smaller subhypergraphs or clusters, in the fashion of hierarchical clustering (see [Das16]). The H-decomposition is a way to represent a dihypergraph as a tree while preserving its vertices and arcs. The notion of a split of a dihypergraph is the principal tool we will use to achieve the H-decomposition. A split of a dihypergraph $\mathcal{H} = (\mathrm{V}, \mathcal{E})$ is a partitioning of the dihypergraph's vertices into two subsets $(\mathrm{V}_1, \mathrm{V}_2)$ such that the arcs of $\mathcal{H}$ are the disjoint union of the arcs of the induced subhypergraphs $\mathcal{H}[\mathrm{V}_1]$, $\mathcal{H}[\mathrm{V}_2]$ and the bipartite dihypergraph $\mathcal{H}[\mathrm{V}_1, \mathrm{V}_2]$, i.e., for any $e = (B, h) \in \mathcal{E}$ intersecting both V_1 and V_2, we have $B \subseteq \mathrm{V}_1$ and $h \in \mathrm{V}_2$ or vice versa. Clearly, there are dihypergraphs that cannot have a split. Our motivation is to study properties of dihypergraphs that can be H-decomposed into trivial dihypergraphs, i.e., hypergraphs with one vertex. The H-decomposition is represented by a full labelled binary trees called H-tree.

An application for our work arises from the decomposition of closure systems, or lattices. In particular, we wish to apply this decomposition to the problem of finding short representations of closure systems [Kha95, DNV19, Wil17]. Splitting lattices or closure systems is an old question and remains an active topic in several areas in mathematics and computer science. Among the common ways to split a lattice are the subdirect decomposition, the duplication (or doubling) of convex sets [BC02, VBDM15a], and other decompositions summarised in [GW99, GW12, Grä11, KVD05]. The former has been early considered by Birkhoff in [Bir44] where his representation theorem *"Every algebra is a subdirect product of its subdirectly irreducible homomorphic images"* is stated. Jipsen and Rose [JR92] summarize many results related to subdirect decomposition and give a list of subdirectly irreducible lattices. From the algorithmic point of view, several works can be found in [GW99, VBDM15b] where closure systems are represented with binary matrices (known as contexts) instead of dihypergraphs. Database theory community has however provided some decomposition schemes for dihypergraphs such as in [DLM92, SS96] or [Lib93], in view of database normalization. Other works on decomposition of dihypergraphs are considered in [BJJ03, GGPR98, AL17, PSSS20], where authors consider cuts or acyclic k-partitioning of dihypergraph. The split operation however is not a cut in general, while we seek for a full decomposition of a dihypergraph which is neither balanced nor acyclic.

In this paper, we present a polynomial time and space algorithm to achieve such a H-decomposition by producing a corresponding H-tree if it exists. However, there are dihypergraphs that cannot be completely decomposed into trivial components. Therefore, we relax this requirement to more indecomposable dihypergraphs called H-factors. This allows to extend the H-decomposition of dihypergraphs to closure systems and lattices.

The paper is structured as follows. In Section 2 we recall some definitions about dihypergraphs. In Section 3 we define the hierarchical decomposition of dihypergraphs, and its representation by a binary labelled tree. We also give a polynomial time and space algorithm to recognise dihypergraphs having a H-decomposition and produces the tree decomposition. Section 4 extends the

H-decomposition to closure systems and provide some properties that can be useful for closure systems classification.

2 Preliminaries

All the objects considered in this paper are finite. For a set V, we denote by 2^{V} its powerset, and for $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, \ldots, n\}$. Sometimes, We omit braces for sets, writing $v_1 v_2 \ldots v_n$ instead of $\{v_1, \ldots, v_n\}$.

We mainly refer to papers [AL17, GLPN93] for terminology and definitions of dihypergraphs. A *directed hypergraph* (*dihypergraph* for short) $\mathcal{H}$ is a pair $(\mathrm{V}(\mathcal{H}), \mathcal{E}(\mathcal{H}))$ where $\mathrm{V}(\mathcal{H})$ is a set of *vertices*, and $\mathcal{E}(\mathcal{H}) = \{e_1, \ldots, e_n\}$, $n \in \mathbb{N}$, a set of *(hyper)arcs*. An arc $e \in \mathcal{E}(\mathcal{H})$ is a pair $(B(e), h(e))$, where $\emptyset \neq B(e) \subseteq \mathrm{V}$ is called the *body* of e and $h(e) \in \mathrm{V} \setminus B$ is the *head* of e. For simplicity, we shall write V, $\mathcal{E}$ and (B, h) for $\mathrm{V}(\mathcal{H})$, $\mathcal{E}(\mathcal{H})$ and $(B(e), h(e))$ respectively. An arc $e = (B, h)$ is written as the set $e = B \cup \{h\}$ when no confusion can arise. If the body B of an arc is reduced to a single element b, we shall write (b, h) instead of $(\{b\}, h)$. In this case, the arc (b, h) is *unit*. If a dihypergraph has only unit edges, it is a *digraph*.

Let $\mathcal{H} = (\mathrm{V}, \mathcal{E})$ be a dihypergraph and U a subset of V. The subhypergraph $\mathcal{H}[U]$ *induced* by U is the pair $(U, \mathcal{E}(\mathcal{H}[U]))$ where $\mathcal{E}(\mathcal{H}[U])$ is the set of arcs of $\mathcal{E}$ contained in U, namely $\mathcal{E}(\mathcal{H}[U]) = \{e \in \mathcal{E} \mid e \subseteq U\}$. A *bipartite dihypergraph* is a dihypergraph in which the ground set can be partitioned into two parts $(\mathrm{V}_1, \mathrm{V}_2)$ such that for any $(B, h) \in \mathcal{E}$, $B \subseteq \mathrm{V}_1$ and $h \in \mathrm{V}_2$ or $B \subseteq \mathrm{V}_2$ and $h \in \mathrm{V}_1$. We denote a bipartite dihypergraph by $\mathcal{H}[\mathrm{V}_1, \mathrm{V}_2]$. The *size* of a dihypergraph $\mathcal{H}$ is written $|\,\mathcal{H}\,|$ and is given by $|\,\mathcal{H}\,| = |\,\mathrm{V}\,| + \sum_{e \in \mathcal{E}} |B(e)| + 1$.

3 Hierarchical decomposition of a dihypergraph

In this section, we introduce a hierarchical decomposition (H-decomposition) of a dihypergraph as a recursive partition of the arcs into bipartite dihypergraphs, from which it can be fully recovered. We are interested first in the class of dihypergraphs that have a hierarchical decomposition. Given a dihypergraph $\mathcal{H} = (\mathrm{V}, \mathcal{E})$, we define the partitioning operation called a *split* of $\mathcal{H}$. Then we recursively apply the splitting operation until reaching trivial dihypergraphs. The H-decomposition of a dihypergraph $\mathcal{H}$ will be represented by a rooted binary tree, called $\mathcal{H}$*-tree*.

We show that not all dihypergraphs can have such a H-decomposition into trivial dihypergraphs, and give a polynomial time and space algorithm which takes a dihypergraph as an input, and outputs a $\mathcal{H}$-tree if it exists. Moreover, we relax the requirement of the H-decomposition into trivial dihypergraphs to H-factors which are indecomposable subhypergraphs of $\mathcal{H}$.

3.1 Split operation

First we define the split operation of a dihypergraph as follows.

Definition 1 (split). *Let* $\mathcal{H} = (\mathrm{V}, \mathcal{E})$ *be a dihypergraph. A non-trivial bipartition* $(\mathrm{V}_1, \mathrm{V}_2)$ *of the groundset* V *is a* split *of* $\mathcal{H}$*, if for any* $e = (B, h) \in \mathcal{E}$, $B \subseteq \mathrm{V}_1$ *or* $B \subseteq \mathrm{V}_2$.

A split $(\mathrm{V}_1, \mathrm{V}_2)$ induces three subhypergraphs $\mathcal{H}[\mathrm{V}_1]$, $\mathcal{H}[\mathrm{V}_2]$ and a bipartite dihypergraph $\mathcal{H}[\mathrm{V}_1, \mathrm{V}_2] = (\mathrm{V}_1, \mathrm{V}_2, \mathcal{E}_{12})$ where $\mathcal{E}_{12} = \{e \in \mathcal{E} \mid e \nsubseteq \mathrm{V}_1 \text{ and } e \nsubseteq \mathrm{V}_2\}$. Moreover, the arcs of $\mathcal{H}[\mathrm{V}_1]$, $\mathcal{H}[\mathrm{V}_2]$ and $\mathcal{H}[\mathrm{V}_1, \mathrm{V}_2]$ form a partition of $\mathcal{E}$. Indeed, no arc is missed by a split. Intuitively, the split shows that $\mathcal{H}$ is fully described by two smaller distincts dihypergraphs $\mathcal{H}[\mathrm{V}_1]$ and $\mathcal{H}[\mathrm{V}_2]$ acting on each other through the bipartite dihypergraph $\mathcal{H}[\mathrm{V}_1, \mathrm{V}_2]$.

Example 1. Consider the dihypergraph $\mathcal{H} = (\mathrm{V}, \mathcal{E})$ of Figure 1, with $\mathrm{V} = [7]$ and $\mathcal{E} = \{(12, 3), (3, 1), (56, 2), (23, 7), (45, 6), (5, 7)\}$. The bipartition illustrated on the left separates V in two sets $\{1, 3\}$ and $\{2, 4, 5, 6, 7\}$. It is not a split since the body of the arc $(12, 3)$ intersects the two parts, and will be missed. The bipartition on the right $\mathrm{V}_1 = \{1, 2, 3\}$ and $\mathrm{V}_2 = \{4, 5, 6, 7\}$ is a split, with $\mathcal{H}[\mathrm{V}_1] = (\{1, 2, 3\}, \{(12, 3), (3, 1)\})$, $\mathcal{H}[\mathrm{V}_2] = (\{1, 2, 3\}, \{(45, 6), (5, 7)\})$, and $\mathcal{H}[\mathrm{V}_1, \mathrm{V}_2] = (\{1, 2, 3\} \cup \{4, 5, 6, 7\}, \{(56, 2), (23, 7)\})$.

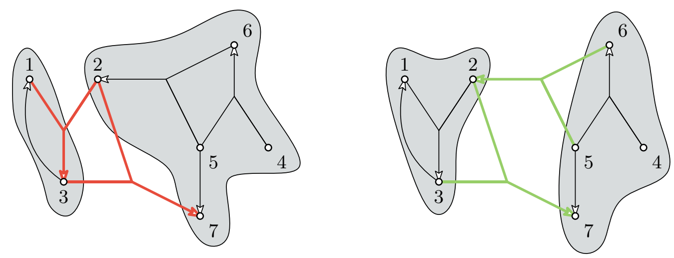

Fig. 1: The left bipartition is a not a split, whereas the right one is.

Before giving a characterization of dihypergraphs having a split, we consider some special cases:

- If the dihypergraph $\mathcal{H}$ is a digraph or has no arc. Then any bipartition of the ground set (any cut) is a split.
- However, there are dihypergraphs that cannot have a bipartition that corresponds to a split. For example, any bipartition of the dihypergraph $\mathcal{H} = (\{1, 2, 3\}, \{(12, 3), (13, 2)\})$ would miss an arc. For instance, if we consider the bipartition $\mathrm{V}_1 = \{1, 2\}$ and $\mathrm{V}_2 = \{3\}$, then we capture $(12, 3)$ but not $(13, 2)$, i.e., $\mathcal{H}[\mathrm{V}_1] = (\{1, 2\}, \emptyset)$, $\mathcal{H}[\mathrm{V}_2] = (\{3\}, \emptyset)$, and $\mathcal{H}[\mathrm{V}_1, \mathrm{V}_2] = (\{1, 2\} \cup \{3\}, \{(12, 3)\})$.

In the following, we show that the dihypergraph's connectivity is important for the notion of a split. Given a dihypergraph $\mathcal{H} = (\mathrm{V}, \mathcal{E})$, we define a *body-path* in $\mathcal{H}$ to be a sequence $v_1, e_1, v_2, ..., v_k, e_k, v_{k+1}$ of distinct vertices and arcs of $\mathcal{H}$ such that: (1) $v_i \in \mathrm{V}, i \in [k+1]$, (2) $e_i = (B_i, h_i) \in \mathcal{E}, i \in [k]$, and (3) $\{v_i, v_{i+1}\} \subseteq B_i, i \in [k]$. Two vertices $v, v' \in \mathrm{V}$ are said to be *body-connected* in $\mathcal{H}$ if there exists a body-path from v and v'. A dihypergraph $\mathcal{H}$ is *body-connected* if every pair of vertices $v, v' \in \mathrm{V}$ is body-connected in $\mathcal{H}$. A body-connected component of a dihypergraph $\mathcal{H}$ is a maximal subset of V where any pair of vertices is body-connected. This is the case for instance of the dihypergraph $\mathcal{H} = (\{1,2,3\}, \{(12,3),(13,2)\})$ we discussed earlier.

Observe that a body reduced to a singleton always satisfies condition of Definition 1. Thus, unit arcs have no impact on a split.

Proposition 1. *A dihypergraph $\mathcal{H}$ has a split iff it is not body-connected.*

Proof. Suppose that $\mathcal{H}$ has a non trivial split $(\mathrm{V}_1, \mathrm{V}_2)$, and let $v \in \mathrm{V}_1$ and $v' \in \mathrm{V}_2$. Assume the existence of a body-path $v = v_1, e_1, v_2, ..., v_k, e_k, v_{k+1} = v'$. Such a body-path exists if there is $i \in [k]$ such that $e_i = (B_i, h_i)$ and $B_i \cap \mathrm{V}_1 \neq \emptyset$ and $B_i \cap \mathrm{V}_2 \neq \emptyset$. But, the arc $e_i = (B_i, h_i)$ cannot satisfy conditions of Definition 1. Then v, v' are not body-connected and thus $\mathcal{H}$ is not body-connected.

Conversely, suppose that $\mathcal{H}$ is not body-connected and let C be a body-connected component of $\mathcal{H}$. We show that $(C, \mathrm{V} \setminus C)$ is a split. Let $e = (B, h) \in \mathcal{E}$. Since C is a body-connected component, either $B \cap C = \emptyset$ or $(\mathrm{V} \setminus C) \cap B = \emptyset$. Hence $(C, \mathrm{V} \setminus C)$ is a split. □

It is important to note that body-connectivity is not inherited. That is, a subhypergraph induced by a body-connected component may not be body-connected. Consider the dihypergraph in Figure 1 with the split $\mathrm{V}_1 = \{1,2,3\}$ and $\mathrm{V}_2 = \{4,5,6,7\}$. Then 5 and 6 are body-connected in $\mathcal{H}$ but not in $\mathcal{H}[\mathrm{V}_2]$. Therefore, body-connected components may be decomposed too. The main idea of the H-decomposition is to recursively apply the split operation until we reach a trivial dihypergraph.

3.2 $\mathcal{H}$-tree of a dihypergraph

Based on the split operation, we define the H-decomposition of a dihypergraph. We recursively split a dihypergraph into smaller dihypergraphs until we reach a trivial dihypergraph. This recursive decomposition can be conveniently represented by a full rooted binary tree T. An interior node of the tree corresponds to a split $(\mathrm{V}_1, \mathrm{V}_2)$ whose children correspond to the H-decomposition of $\mathcal{H}[\mathrm{V}_1]$ and $\mathcal{H}[\mathrm{V}_2]$. The leaves of the tree represent the ground set. Since the splits $(\mathrm{V}_1, \mathrm{V}_2)$ and $(\mathrm{V}_2, \mathrm{V}_1)$ are equivalent, the children of a node are not ordered.

Definition 2 ($\mathcal{H}$-tree). *Let $\mathcal{H} = (\mathrm{V}, \mathcal{E})$ be a dihypergraph, T be a full rooted binary tree. Then (T, λ) is a $\mathcal{H}$-tree of $\mathcal{H}$ if there exists a labelling map $\lambda \colon T \to \mathrm{V} \cup 2^{\mathcal{E}}$ satisfying the following conditions:*

(i) $\lambda(v) \in \mathrm{V}$ *if v is a leaf of T,*
(ii) $\lambda(v) \subseteq \mathcal{E}$ *if v is an interior node (possibly $\lambda(v) = \emptyset$),*
(iii) for any $(B, h) \in \lambda(v)$, elements of B are labels of leaves in the subtree of one child of v and h is the label of a leaf in the subtree of the other child.

(iv) the set $\{\lambda(v) \mid v \in T\}$ *is a full partition of* $\mathrm{V} \cup \mathcal{E}$ *and may contain the emptyset.*

If such labelling exists we call the dihypergraph hierarchically decomposable (H-decomposable for short), and H-indecomposable otherwise.

Figure 2 shows a $\mathcal{H}$-tree for the dihypergraph in Figure 1.

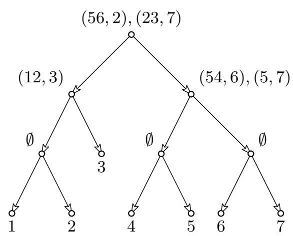

Fig. 2: A H-decomposition for the dihypergraph in Figure 1

There are two interesting cases where a H-decomposition of a dihypergraph $\mathcal{H}$ can be computed easily (see Figure 3).

- $\mathcal{H}$ has no arcs. Here, any full rooted binary tree whose leaves are labelled by a permutation of V and any interior node by $\emptyset$ is a $\mathcal{H}$-tree of $\mathcal{H}$.
- $\mathcal{H}$ is a *digraph*. It is the same as the previous case, except that an arc (b, h) will be in the label of the ancestor of the leaves labelled by b and h.

However, there are also some dihypergraphs that cannot be H-decomposed.

Fig. 3: Hierarchical decompositions for the empty dihypergraphs $\mathcal{H}_1 = ([4], \emptyset)$, and for the directed graph $\mathcal{H}_2 = ([4], \{(1, 2), (2, 3), (3, 4), (4, 1)\})$

Proposition 2. *If* $\mathcal{H}$ *is H-decomposable then it is not body-connected.*

Proof. Suppose that $\mathcal{H}$ is H-decomposable, and let (T, λ) be a $\mathcal{H}$-tree with root r. Let $(\mathrm{V}_l, \mathrm{V}_r)$ be the split of V corresponding to r, i.e., V_l corresponds to the leaves of the left subtree of r and V_r to those of the right subtree. Then according to Proposition 1, $\mathcal{H}$ is not body-connected. $\square$

Now, we show that H-decomposability is hereditary, i.e., if a dihypergraph $\mathcal{H}$ has a $\mathcal{H}$-tree then any of its subhypergraphs has a H-decomposition too.

Proposition 3. *Let* $\mathcal{H} = (\mathrm{V}, \mathcal{E})$ *be a dihypergraph and* $U \subseteq \mathrm{V}$*. If* $\mathcal{H}$ *is H-decomposable, so is* $\mathcal{H}[U]$*.*

Proof. Let $\mathcal{H} = (\mathrm{V}, \mathcal{E})$ be a dihypergraph, $U \subseteq \mathrm{V}$ and (T, λ) a $\mathcal{H}$-tree. We construct a subtree not necessarily induced by T which corresponds to a $\mathcal{H}[U]$-tree. We start from the root r of T and apply the following operation for any interior node v: if the sets of leaves of the left child and those of the right one intersect both U, then keep v with label $\lambda(v) = \lambda(v) \cap \mathcal{E}(\mathcal{H}[U])$. Otherwise, there is a child of v whose set of leaves do not intersect U, in this case replace v by the child whose set of leaves intersects U. The obtained subtree has U as the set of leaves, and the set of labels of the internal nodes are exactly $\mathcal{E}(\mathcal{H}[U])$. $\square$

The following theorem gives the strategy of the algorithm for recognizing which hypergraphs have a H-decomposition.

Theorem 1. *Let $\mathcal{H} = (\mathrm{V}, \mathcal{E})$ be a non body-connected dihypergraph and C a body-connected component of $\mathcal{H}$. Then $\mathcal{H}$ is H-decomposable if and only if both $\mathcal{H}[C]$ and $\mathcal{H}[\mathrm{V} \setminus C]$ are H-decomposable.*

Proof. The only if part directly follows from Proposition 3. Let us show the if part. Let C be a body-connected component of $\mathcal{H}$ and let (T_1, λ_1) be a $\mathcal{H}[C]$-tree and (T_2, λ_2) a $\mathcal{H}[\mathrm{V} \setminus C]$-tree. We consider a new tree (T, λ) such that T has root r with left subtree T_1 and right subtree T_2. As for λ, we put $\lambda(v) = \lambda_1(v)$ if $v \in T_1$, $\lambda(v) = \lambda_2(v)$ if $v \in T_2$ and $\lambda(r) = \{e \in \mathcal{E} \mid e \notin \mathcal{E}(\mathcal{H}[C]) \cup \mathcal{E}(\mathcal{H}[\mathrm{V} \setminus C])\}$. In words, $\lambda(r)$ contains any arc which is not fully contained in C or $\mathrm{V} \setminus C$. It is clear that conditions *(i), (ii), (iv)* of Definition 2 are fulfilled for (T, λ) as they are for (T_1, λ_1), (T_2, λ_2) and $C \cup \mathrm{V} \setminus C = \mathrm{V}$. Hence, we have to check *(iii)*. Let $e = (B, h)$ be an arc in $\lambda(v)$. If $B \cap C \neq \emptyset$, then $B \subseteq C$ since C is a body-connected component of $\mathcal{H}$. As e is not an arc of $\mathcal{H}[C]$, it follows that $h \in \mathrm{V} \setminus C$. Dually, if $B \cap C = \emptyset$, then $h \in C$ since e is not in $\mathcal{H}[\mathrm{V} \setminus C]$. Condition *(iii)* is satisfied and (T, λ) is a $\mathcal{H}$-tree. $\square$

Theorem 1 suggests a recursive algorithm which returns a $\mathcal{H}$-tree for $\mathcal{H}$ if it is H-decomposable. If $\mathcal{H}$ is reduced to a vertex v, we output a tree which is a leaf with label v. Otherwise, we compute a body-connected component C of $\mathcal{H}$ if $\mathcal{H}$ is not body-connected. We label the corresponding node by the arcs of $\mathcal{H}[C, \mathrm{V} \setminus C]$, and we recursively call the algorithm on $\mathcal{H}[C]$ and $\mathcal{H}[\mathrm{V} \setminus C]$.

This strategy is formalized in Algorithm 1, whose correctness and complexity are studied in Theorem 2.

Theorem 2. *Given a dihypergraph $\mathcal{H} = (\mathrm{V}, \mathcal{E})$, Algorithm* `BuildTree` *computes a $\mathcal{H}$-tree if it exists and returns* FAIL *otherwise in polynomial time and space in the size of $\mathcal{H}$.*

Proof. First, we show by induction on $|\mathrm{V}|$ that the algorithm returns a $\mathcal{H}$-tree iff $\mathcal{H}$ is H-decomposable by. Clearly, if $\mathcal{H}$ is reduced to a vertex v, the algorithm returns a $\mathcal{H}$-tree corresponding to a leaf with label v.

Now, assume that the algorithm is correct for dihypergraphs with $|\mathrm{V}| < n$, $n \in \mathbb{N}$, and consider a dihypergraph $\mathcal{H}$ with $|\mathrm{V}| = n$. Suppose $\mathcal{H}$ is H-decomposable. By Proposition 1, $\mathcal{H}$ is not body-connected. Let C be a body-connected component of $\mathcal{H}$. Inductively, the algorithm is correct for $\mathcal{H}[C]$ and

Algorithm 1: `BuildTree`

Input: A dihypergraph $\mathcal{H} = (\mathrm{V}, \mathcal{E})$
Output: A $\mathcal{H}$-tree, if it exists, FAIL otherwise

1 **if** $\mathcal{H}$ *has one vertex* **then**
2   create a new leaf r with $\lambda(r)$ the unique vertex in $\mathcal{H}$;
3   return r ;
4 **else**
5   compute a body-connected component C of $\mathcal{H}$;
6   **if** $|C| = |\mathrm{V}|$ **then**
7   stop and return FAIL ;
8   **else**
9   let r be a new node with $\lambda(r) = \mathcal{E} \setminus (\mathcal{E}(\mathcal{H}[C]) \cup \mathcal{E}(\mathcal{H}[\mathrm{V} \setminus C]))$;
10   left$(r) =$ `BuildTree`$(\mathcal{H}[C])$;
11   right$(r) =$ `BuildTree`$(\mathcal{H}[\mathrm{V} \setminus C])$;
12   return r ;

$\mathcal{H}[\mathrm{V} \setminus C]$ since $1 \leq |C| < n$. From Theorem 1, we have that both $\mathcal{H}[C]$ and $\mathcal{H}[\mathrm{V} \setminus C]$ are H-decomposable. By induction the algorithm computes a $\mathcal{H}[C]$-tree (T_1, λ_1) and a $\mathcal{H}[\mathrm{V} \setminus C]$-tree (T_2, λ_2). Hence, the algorithm returns a labelled tree (T, λ) with root r whose label is $\lambda(r) = \mathcal{E} \setminus (\mathcal{E}(\mathcal{H}[C]) \cup \mathcal{E}(\mathcal{H}[\mathrm{V} \setminus C]))$ and children T_1 and T_2. This tree satisfies all conditions to be a $\mathcal{H}$-tree. Thus the algorithm computes a $\mathcal{H}$-tree for any H-decomposable dihypergraph.

Now suppose $\mathcal{H}$ is not H-decomposable. We have two cases:

1. $\mathcal{H}$ is body-connected and the algorithm returns FAIL in Line **7**.
2. $\mathcal{H}$ is not body-connected. The algorithm chooses a body-connected component C with $1 \leq |C| < n$. By Theorem 1, either $\mathcal{H}[C]$ or $\mathcal{H}[\mathrm{V} \setminus C]$ is H-indecomposable. Thus by induction, the algorithm will return FAIL for the input $\mathcal{H}[C]$ or $\mathcal{H}[\mathrm{V} \setminus C]$ in Lines **11-12**. Since the algorithm stops, the output of the algorithm is FAIL.

Hence, the algorithm fails if the input dihypergraph $\mathcal{H}$ is H-indecomposable. We conclude that the algorithm returns a $\mathcal{H}$-tree if and only if the input dihypergraph $\mathcal{H}$ is H-decomposable.

Now we show that the total time and space complexity of the algorithm are polynomial. The space required for the algorithm is bounded by the size of the dihypergraph and the size of the $\mathcal{H}$-tree. As the size of the $\mathcal{H}$-tree is bounded by $O(|\mathcal{H}|)$, the overall space is bounded by $O(|\mathcal{H}|)$.

The time complexity is bounded by the sum of the costs of all nodes (or calls) of the search tree. The number of calls is bounded by $O(|\mathrm{V}|)$, the size of the search tree. The cost of a call is dominated by the computation of a body-connected component of the input $\mathcal{H}$. For this, we use union-find data structure of [TR84], which runs in almost linear time, i.e., $O(|\mathcal{H}| \cdot \alpha(|\mathcal{H}|, |\mathrm{V}|))$ where $\alpha(.,.)$ is the inverse Ackermann function. The almost linear comes from the fact that $\alpha(|\mathrm{V}|) \leq 4$ for any practical dihypergraph (see [TR84]). Thus the total time complexity is $O(|\mathrm{V}|(|\mathcal{H}| \cdot \alpha(|\mathcal{H}|, |\mathrm{V}|))$. □

It is worth noticing, that the obtained $\mathcal{H}$-tree by Algorithm 1 depends on the choice of a body-connected component in line **5**. Thus, there may be several $\mathcal{H}$-trees for a given dihypergraph. Figure 4 shows two possible $\mathcal{H}$-trees for the dihypergraph $\mathcal{H} = (\mathrm{V}, \mathcal{E})$ with $\mathrm{V} = [8]$ and $\mathcal{E} = \{(12,3),(23,4),(34,5),(56,7),(67,8)\}$. Then, a question arises: are all *$\mathcal{H}$-trees equivalently interesting?* In particular, a balanced $\mathcal{H}$-tree is a good candidate as the balancing is a common desirable property for decomposition trees to obtain efficient algorithms.

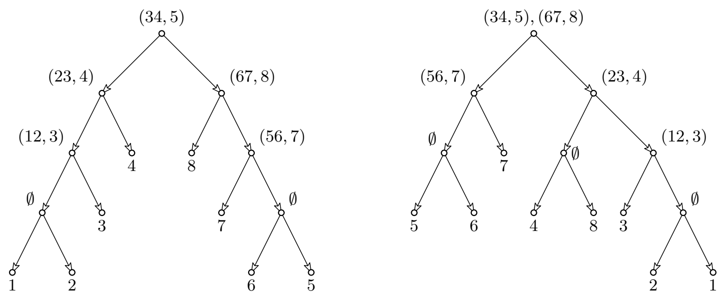

Fig. 4: Two possible $\mathcal{H}$-trees of the same dihypergraph

3.3 Extension of the H-decomposition

As seen before, there are dihypergraphs that cannot have a split and thus a H-decomposition into trivial hypergraphs. Such dihypergraphs are body-connected, and will be called *irreducible H-factors* (H-factors for short) in the rest of the paper. Now we describe a slight modification of Algorithm 1 to obtain a H-decomposition of dihypergraphs into H-factors. Instead of returning FAIL in line **7** in Algorithm `BuildTree`, we replace it by the following:

7' create a new leaf r with $\lambda(r) = \mathcal{E}$ and return r;

Figure 5 illustrates the H-decomposition of a dihypergraph, where the leftmost leaf corresponds to a non-trivial H-factor. Now, any dihypergraph has a H-decomposition into H-factors. Then, the decomposition can be applied to any objects encoded by dihypergraphs, as we will show for closure systems in the next section.

4 H-decomposition of a closure system into H-factors

In this section, we show that the H-decomposition introduced in the previous section can be applied to decompose closure systems when they are represented by dihypergraphs [Wil17, AL17].

We first recall definitions for closure systems and lattice theory [Grä11]. A partially ordered set $\mathcal{L} = (L, \leq)$ is a reflexive, anti-symmetric and transitive binary relation $\leq$ on a set L. For $x, y \in \mathcal{L}$, we say that x and y are *comparable* if $x \leq y$ or $y \leq x$. An upper bound of x, y is an element $u \in \mathcal{L}$ such that $x \leq u$,

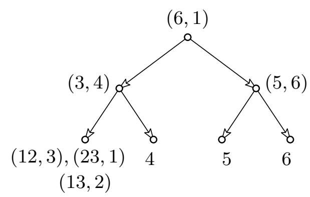

Fig. 5: H-decomposition into H-factors

$y \leq u$. If for any upper bound $u' \neq u$, $u \leq u'$, then u is the *join* of x, y, written $x \vee y$. Lower bounds and the meet $x \wedge y$ are defined dually. We say that $\mathcal{L}$ is a *lattice* if for any $x, y \in \mathcal{L}$, both $x \vee y$ and $x \wedge y$ exist. A *meet-sublattice* $\mathcal{L}'$ of $\mathcal{L}$ is a lattice included in $\mathcal{L}$ such that for any $x, y \in \mathcal{L}'$, $x \wedge y \in \mathcal{L}'$. It is a *sublattice* of $\mathcal{L}$ if moreover $x \vee y \in \mathcal{L}'$.

A closure system $\mathcal{F}$ on a finite set V is a family of subsets of V which contains V and is closed under intersection, that is for any F_1, F_2 in the family $\mathcal{F}$, $F_1 \cap F_2$ also belongs to $\mathcal{F}$. A subset F of V which is in $\mathcal{F}$ is called a *closed set*. It is well known that a closure system ordered by set containment is a lattice and that any lattice can be associated to a closure system [Grä11]. The *projection* or *trace* of a closure system $\mathcal{F}$ over $U \subseteq \mathrm{V}$, denoted $\mathcal{F} \colon U$, is the closure system we obtain by intersecting each $F \in \mathcal{F}$ with U, i.e., $\mathcal{F} \colon U = \{F \cap U \mid F \in \mathcal{F}\}$. The direct product of two closure systems $\mathcal{F}_1, \mathcal{F}_2$ on disjoint $\mathrm{V}_1, \mathrm{V}_2$ is the pairwise union of their closed sets, that is $\mathcal{F}_1 \times \mathcal{F}_2 = \{F_1 \cup F_2 \mid F_1 \in \mathcal{F}_1, F_2 \in \mathcal{F}_2\}$.

To compute the closure system associated to a dihypergraph $\mathcal{H}$, we recall the *forward chaining* method. Let $\mathcal{H}$ be a dihypergraph and $X \subseteq \mathrm{V}$, we construct a chain of subsets of V, $X = X_0 \subset X_1 \subset ... \subset X_k = X^{\mathcal{H}}$, where $X_i = X_{i-1} \cup \{h \mid B \subseteq X_{i-1}, (B, h) \in \mathcal{E}\}$ with $i > 0$. The subset $X^{\mathcal{H}}$ is called a closed set. Note that a subset X of V is closed if for any arc $(B, h), B \subseteq X$ implies $h \in X$. The family of closed sets $\mathcal{F}_{\mathcal{H}} = \{X^{\mathcal{H}} \mid X \subseteq \mathrm{V}\}$ is a closure system. A closure system can be represented by several dihypergraphs.

Naturally, we wish to extend the H-decomposition of a dihypergraph $\mathcal{H}$ to a decomposition of the closure system $\mathcal{F}_{\mathcal{H}}$, also called H-decomposition. The H-decomposition of $\mathcal{F}_{\mathcal{H}}$ is obtained from the H-decomposition of $\mathcal{H}$, where the label of a node of its $\mathcal{H}$-tree is replaced by the closure system associated to the dihypergraph induced by its subtree. The closure systems corresponding to leaves are the irreducible H-factors of the input closure system. Figure 6 illustrates the H-decomposition of the closure system associated to the H-decomposition of the dihypergraph in Figure 5.

Theorem 3. *Let* $(\mathrm{V}_1, \mathrm{V}_2)$ *be a split of* $\mathcal{H}$, $\mathcal{F}_1$ *and* $\mathcal{F}_2$ *the closure systems corresponding to* $\mathcal{H}[\mathrm{V}_1]$ *and* $\mathcal{H}[\mathrm{V}_2]$ *respectively. Then,*

1. *If* $F \in \mathcal{F}_{\mathcal{H}}$ *then* $F_i = F \cap \mathrm{V}_i \in \mathcal{F}_i, i = \{1, 2\}$. *Moreover,* $\mathcal{F}_{\mathcal{H}} \subseteq \mathcal{F}_1 \times \mathcal{F}_2$.
2. *If* $\mathcal{H}[\mathrm{V}_1, \mathrm{V}_2]$ *has no arc then* $\mathcal{F}_{\mathcal{H}} = \mathcal{F}_1 \times \mathcal{F}_2$.
3. *If* $B \subseteq \mathrm{V}_1$ *for any arc* (B, h) $\mathcal{H}[\mathrm{V}_1, \mathrm{V}_2]$, *then* $\mathcal{F}_{\mathcal{H}} : \mathrm{V}_i = \mathcal{F}_i$ *for* $i \in \{1, 2\}$.
4. *If* $B \subseteq \mathrm{V}_2$ *for any arc* (B, h) *of* $\mathcal{H}[\mathrm{V}_1, \mathrm{V}_2]$, *then* $\mathcal{F}_{\mathcal{H}} : \mathrm{V}_i = \mathcal{F}_i$ *for* $i \in \{1, 2\}$.

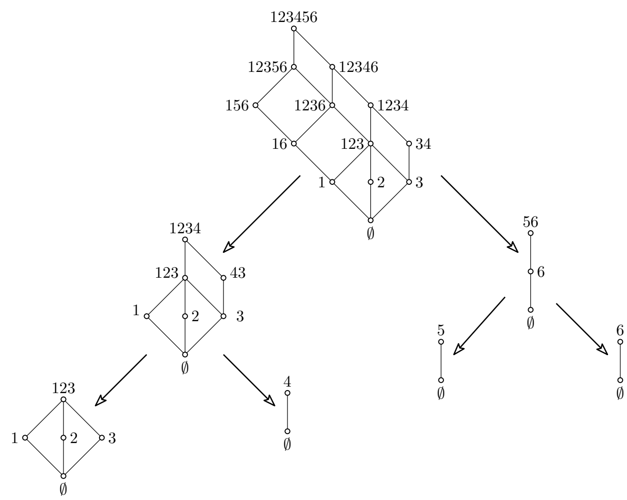

Fig. 6: The H-decomposition of the closure system corresponding to the dihypergraph in Figure 5

Proof. Consider a split $(\mathrm{V}_1, \mathrm{V}_2)$ of $\mathcal{H}$, $\mathcal{F}_1$ and $\mathcal{F}_2$ the closure systems corresponding to $\mathcal{H}[\mathrm{V}_1]$ and $\mathcal{H}[\mathrm{V}_2]$. We will prove *1*, *3* and *2*. Items *3*, *4* are similar.

1. Let $F \in \mathcal{F}_{\mathcal{H}}$ with $F_i = F \cap \mathrm{V}_i$ and (B, h) an arc of $\mathcal{H}[\mathrm{V}_i]$ for $i \in \{1, 2\}$. Suppose $B \subseteq F_i$ and $h \notin F_i$. Then we also have $B \subseteq F$ and $h \notin F$ which contradicts $F \in \mathcal{F}_{\mathcal{H}}$, as (B, h) is an arc of $\mathcal{H}$.

3. Let $F \in \mathcal{F}_1$, we show that $F^{\mathcal{H}}$ satisfies $F^{\mathcal{H}} \cap \mathrm{V}_1 = F$. Let (B, h) an arc of $\mathcal{H}$. We distinguish 3 cases:
   (a) $B \subseteq \mathrm{V}_2$. Then $B \not\subseteq F$ and (B, h) has no effect on the forward chaining.
   (b) (B, h) is in $\mathcal{H}[\mathrm{V}_1]$. Then $B \subseteq F$ implies $h \in F$ as it is closed in $\mathcal{H}[\mathrm{V}_1]$.
   (c) (B, h) is an arc of $\mathcal{H}[\mathrm{V}_1, \mathrm{V}_2]$. Then $B \subseteq \mathrm{V}_1$ and $h \in \mathrm{V}_2$. As there is no arc outside $\mathcal{H}[\mathrm{V}_1]$ with head in V_1 the forward chaining cannot add an element from V_1.

   So $F \in \mathcal{F}_{\mathcal{H}} \colon \mathrm{V}_1$ and $\mathcal{F}_1 \subseteq \mathcal{F} \colon \mathrm{V}_1$. The reverse inclusion holds by item *1*. As for $\mathcal{F}_2$, we have $\mathcal{F}_2 \subseteq \mathcal{F}$ as $B \subseteq \mathrm{V}_1$ for any arc (B, h) of $\mathcal{H}[\mathrm{V}_1, \mathrm{V}_2]$.

2. Since $\mathcal{H}$ has no arc, it satisfies *3* and *4*. We deduce that $\mathcal{F}_{\mathcal{H}} \colon \mathrm{V}_1 = \mathcal{F}_1$ and $\mathcal{F}_{\mathcal{H}} \colon \mathrm{V}_2 = \mathcal{F}_2$. Thus $\mathcal{F}_{\mathcal{H}} \subseteq \mathcal{F}_1 \times \mathcal{F}_2$. For the other inclusion, let $F_1 \in \mathcal{F}_1$ and $F_2 \in \mathcal{F}_2$. We show that $F_1 \cup F_2 \in \mathcal{F}_{\mathcal{H}}$. Let (B, h) be an arc of $\mathcal{H}$ with $B \subseteq F_1 \cup F_2$. As $\mathcal{H}[\mathrm{V}_1, \mathrm{V}_2]$ has no arc, (B, h) is an arc of $\mathcal{H}[\mathrm{V}_1]$ or $\mathcal{H}[\mathrm{V}_2]$. As F_1, F_2 are closed for $\mathcal{H}[\mathrm{V}_1]$, $\mathcal{H}[\mathrm{V}_2]$ (resp.), it follows that $F_1 \cup F_2 \in \mathcal{F}_{\mathcal{H}}$. □

According to Theorem 3 *1*, any closure system is a subset of the product of its H-factors closure systems. So the idea is to compute in parallel $\mathcal{F}_1$ and $\mathcal{F}_2$ for every split $(\mathrm{V}_1, \mathrm{V}_2)$ in the $\mathcal{H}$-tree, and then use the dihypergraph $\mathcal{H}[\mathrm{V}_1, \mathrm{V}_2]$ to compute $\mathcal{F}_{\mathcal{H}}$. But this strategy is expensive, since the size of $\mathcal{F}_1$ and $\mathcal{F}_2$ may be exponential in the size of $\mathcal{F}_{\mathcal{H}}$. Indeed, take $\mathcal{H}$ with $\mathrm{V} = \{u_1, \ldots, u_n, z\}$ and

$\mathcal{E} = \bigcup_{i=1}^{n} \{(u_i, z), (z, u_i)\}$. Then, $(\mathrm{V} \setminus \{z\}, \{z\})$ is a split of $\mathcal{H}$ with $\mathcal{H}[\mathrm{V} \setminus \{z\}] = (\mathrm{V} \setminus \{z\}, \emptyset)$. Hence $\mathcal{F}_{\mathrm{V} \setminus \{z\}} = 2^{\mathrm{V} \setminus \{z\}}$ has exponential size, while $\mathcal{F}_{\mathcal{H}} = \{\emptyset, \mathrm{V}\}$. Observe that this explosion cannot happen if $\mathcal{F}_1$ and $\mathcal{F}_2$ are traces of $\mathcal{F}_{\mathcal{H}}$.

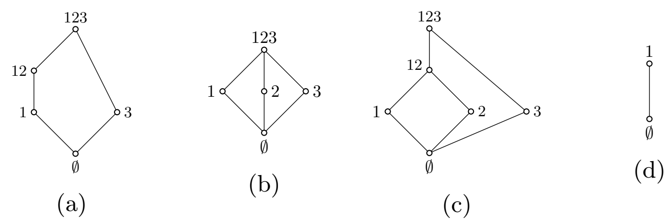

Fig. 7

To conclude this section, we relate H-decomposition to the subdirect product decomposition. Consider the closure system $\mathcal{F}_{\mathcal{H}}$ in Figure 7(a) encoded by the unique dihypergraph $\mathcal{H} = (\{1, 2, 3\}, \{(2, 1), (13, 2)\})$. It is known that it cannot be decomposed using the subdirect product. Clearly $\mathcal{H}$ is not body-connected and $\mathrm{V}_1 = \{1, 3\}$ et $\mathrm{V}_2 = \{2\}$ is the unique split where $\mathcal{F}_1 = \{\emptyset, 1, 3, 13\}$ and $\mathcal{F}_2 = \{\emptyset, 2\}$ are traces. But $\mathcal{F}_{\mathcal{H}}$ is not a sublattice of $\mathcal{F}_1 \times \mathcal{F}_2$, since $(1, 3) \in \mathcal{F}_1 \times \mathcal{F}_2$ the upper bound of 1 and 3 is not preserved in $\mathcal{F}_{\mathcal{H}}$. However, systems of Figure 7(b), (c) and (d) are subdirectly irreducible and H-factors too. Hence, we end the section with the following.

Corollary 1. *Every closure system is a meet-sublattice of the direct product of its H-factors.*

Proof. This follows from Theorem 3 *(i)* and the fact that a closure system is closed under intersection. □

5 Conclusion

In this paper we studied a hierarchical decomposition for dihypergraphs based on a partitioning operation called a split. We gave a polynomial time and space algorithm to find such a decomposition if it exists. We also discussed application of this decomposition to closure systems and lattices.

For future research, we are interested in characterizing classes of closure systems which have a hierarchical decomposition. It seems that it is already the case for lower-bounded lattices, which include the widely studied Tamari lattices [Gey94, Mar92]. Another future application is the translation between representations of closure systems [Kha95, DNV19].

Acknowledgment Authors are grateful to the anonymous reviewers for their remarks and suggestions.

References

[ADS86] Giorgio Ausiello, Alessandro D'Atri, and Domenico Sacca. Minimal representation of directed hypergraphs. *SIAM Journal on Computing*, 15(2):418–431, 1986.

[AL17] Giorgio Ausiello and Luigi Laura. Directed hypergraphs: Introduction and fundamental algorithms—a survey. *Theoretical Computer Science*, 658:293–306, 2017.

[BC02] Karell Bertet and Nathalie Caspard. Doubling convex sets in lattices: characterizations and recognition algorithms. *Order*, 19(2):181–207, 2002.

[BDVG18] Karell Bertet, Christophe Demko, Jean-François Viaud, and Clément Guérin. Lattices, closures systems and implication bases: A survey of structural aspects and algorithms. *Theoretical Computer Science*, 743:93–109, 2018.

[Bir44] Garreth Birkhoff. Subdirect products in universal algebra. *Bull. Amer. Math. Soc*, 50:767–768, 1944.

[BJJ03] Alex R Berg, Bill Jackson, and Tibor Jordán. Edge splitting and connectivity augmentation in directed hypergraphs. *Discrete mathematics*, 273(1-3):71–84, 2003.

[Das16] Sanjoy Dasgupta. A cost function for similarity-based hierarchical clustering. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 118–127, 2016.

[DLM92] János Demetrovics, Leonid Libkin, and Ilya B Muchnik. Functional dependencies in relational databases: a lattice point of view. *Discrete Applied Mathematics*, 40(2):155–185, 1992.

[DNV19] Oscar Defrain, Lhouari Nourine, and Simon Vilmin. Translating between the representations of a ranked convex geometry. *CoRR*, abs/1907.09433, 2019.

[Gey94] Winfried Geyer. On tamari lattices. *Discrete Mathematics*, 133(1-3):99–122, 1994.

[GGPR98] Giorgio Gallo, Claudio Gentile, Daniele Pretolani, and Gabriella Rago. Max horn sat and the minimum cut problem in directed hypergraphs. *Mathematical Programming*, 80(2):213–237, 1998.

[GLPN93] Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete applied mathematics*, 42(2-3):177–201, 1993.

[Grä11] George Grätzer. *Lattice theory: foundation*. Springer Science & Business Media, 2011.

[GW99] Bernhard Ganter and Rudolf Wille. Decompositions of concept lattices. In *Formal Concept Analysis*, pages 129–181. Springer, 1999.

[GW12] Bernhard Ganter and Rudolf Wille. *Formal concept analysis: mathematical foundations*. Springer Science & Business Media, 2012.

[JR92] Peter Jipsen and Henry Rose. Varieties of lattices, volume 1533 of lecture notes in mathematics, 1992.

[Kha95] Roni Khardon. Translating between horn representations and their characteristic models. *Journal of Artificial Intelligence Research*, 3:349–372, 1995.

[KVD05] Jean François Djoufak Kengue, Petko Valtchev, and Clémentin Tayou Djamegni. A parallel algorithm for lattice construction. In *International Conference on Formal Concept Analysis*, pages 249–264. Springer, 2005.

[Lib93] Leonid Libkin. Direct product decompositions of lattices, closures and relation schemes. *Discrete Mathematics*, 112(1-3):119–138, 1993.

[Mar92] George Markowsky. Primes, irreducibles and extremal lattices. *Order*, 9(3):265–290, 1992.

[PSSS20] Merten Popp, Sebastian Schlag, Christian Schulz, and Daniel Seemaier. Multilevel acyclic hypergraph partitioning. *arXiv preprint arXiv:2002.02962*, 2020.

[SS96] Hossein Saiedian and Thomas Spencer. An efficient algorithm to compute the candidate keys of a relational database schema. *The Computer Journal*, 39(2):124–132, 1996.

[TR84] van Leeuwen J. Tarjan RE. Worst-case analysis of set union algorithms. *Journal of the ACM*, 2:245–281, 1984.

[VBDM15a] Jean-François Viaud, Karell Bertet, Christophe Demko, and Rokia Missaoui. The reverse doubling construction. In *2015 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K)*, volume 1, pages 350–357. IEEE, 2015.

[VBDM15b] Jean-François Viaud, Karell Bertet, Christophe Demko, and Rokia Missaoui. Subdirect decomposition of contexts into subdirectly irreducible factors. *Formal Concept Analysis and Applications FCA&A 2015*, page 49, 2015.

[Wil17] Marcel Wild. The joy of implications, aka pure horn formulas: mainly a survey. *Theoretical Computer Science*, 658:264–292, 2017.