

Hierarchical decompositions of dihypergraphs

Lhouari Nourine, Simon Vilmin

▶ **To cite this version:**

Lhouari Nourine, Simon Vilmin. Hierarchical decompositions of dihypergraphs. 2020. hal-02876740

HAL Id: hal-02876740
https://uca.hal.science/hal-02876740

Preprint submitted on 21 Jun 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Hierarchical decompositions of dihypergraphs ⋆

Lhouari Nourine[1] and Simon Vilmin[1]

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 trees 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 $\mathcal{H}$-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)edges (sometimes called hyperarcs) of the form (B, h) over V, where B is a subset and h a singleton of V. In database theory, V corresponds to a relation schema and edges are functional dependencies; whereas in Horn logic an edge is definite Horn clause on the propositional variables set V. In general, an edge (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 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

⋆ The second author is funded by the CNRS, France, ProFan project.

way to represent a dihypergraph as a tree while preserving its vertices and edges. 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 $\mathcal{H} = (V, \mathcal{E})$ is a partitioning of the dihypergraph's vertices into two subset (V_1, V_2) such that the edges of $\mathcal{H}$ are the disjoint union of the edges of the induced subhypergraphs $\mathcal{H}[V_1]$, $\mathcal{H}[V_2]$ and the bipartite dihypergraph $\mathcal{H}[V_1, V_2]$, i.e. for any $e = (B, h) \in \mathcal{E}$ intersecting both V_1 and V_2, we have $B \subseteq V_1$ and $h \in V_2$ 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. The concept of 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 summarised decomposition 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], but these works differ in aims and methods from our work.

In this paper, we present a polynomial time and space algorithm to achieve such a H-decomposition by producing its 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 relaxation allows us to extend the H-decomposition of dihypergraphs to closure systems and lattices. This approach of H-decomposing closure systems permit a deep understanding of the subdirect product via the dihypergraphs representation of closure systems.

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 $\{1, \dots, n\}$. We also sometimes omit braces for sets, writing $v_1 v_2 \dots v_n$ for the set $\{v_1, \dots, v_n\}$.

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 its set of vertices, and $\mathcal{E}(\mathcal{H}) = \{e_1, \dots, e_n\}$, $n \in \mathbb{N}$, its set of *edges*. An edge $e \in \mathcal{E}(\mathcal{H})$ is a pair $(B(e), h(e))$, where $B(e) \subseteq \mathrm{V}$ called the *body* of e and $h(e) \in \mathrm{V} \setminus B$ called the *head* of e.

When it is clear from the context, we write V, $\mathcal{E}$ and (B, h) instead of $\mathrm{V}(\mathcal{H})$, $\mathcal{E}(\mathcal{H})$ and $(B(e), h(e))$ respectively. An edge $e = (B, h)$ is written as the set $e = B \cup \{h\}$ when no confusion can arise. Whenever the body B of an edge is reduced to a single element b, we shall write (b, h) instead of $(\{b\}, h)$ for clarity. In this case, the edge (b, h) is called a *unit edge*. If all the edges of a dihypergraph are unit, then it is called 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 edges 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 $(B, h) \in \mathcal{E}$, $B \subseteq \mathrm{V}_1$ or $B \subseteq \mathrm{V}_2$. 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 $|\mathcal{H}| = |\mathrm{V}| + \sum_{e \in \mathcal{E}} |B(e)| + 1$. The number of edges in $\mathcal{E}$ is written $|\mathcal{E}|$.

Let $T = (\mathrm{V}(T), \mathcal{E}(T))$ be a full rooted binary tree and $v \in \mathrm{V}(T)$. We denote by $\mathsf{left}(v)$ its left child and $\mathsf{right}(v)$ its right one. The subtree induced by v is written $T[v]$, and the leaves of $T[v]$ are given by $\mathsf{leaves}(v)$. Sometimes, we will write $v \in T$ as a shortcut for $v \in \mathrm{V}(T)$. We assume that the ground set $\mathrm{V}(T)$ is disjoint from the ground set of any dihypergraph we will deal with.

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 edges 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 body-connected dihypergraphs.

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 $\mathcal{E}_{12} = \{e \in \mathcal{E} \mid e \not\subseteq \mathrm{V}_1 \text{ and } e \not\subseteq \mathrm{V}_2\}$. Moreover, the edges of $\mathcal{H}[\mathrm{V}_1]$, $\mathcal{H}[\mathrm{V}_2]$ and $\mathcal{H}[\mathrm{V}_1, \mathrm{V}_2]$ form a partition of the edges of $\mathcal{H}$. Indeed, no edge 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 $\mathcal{H} = (\mathrm{V}, \mathcal{E})$ depicted in Figure 1, with $\mathrm{V} = [7]$ and $\mathcal{E} = \{(12, 3), (3, 1), (56, 2), (23, 7), (45, 6), (5, 7)\}$. The bipartition illustrated by the full line separates V in two sets $\{1, 3\}$ and $\{2, 4, 5, 6, 7\}$. It is not a split since the body of the edge $(12, 3)$ intersects the two parts, and will be missed. The bipartition corresponding to the dotted line $\mathrm{V}_1 = \{1, 2, 3\}$ 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 $\mathcal{H}[\mathrm{V}_1, \mathrm{V}_2] = (\{1, 2, 3\} \cup \{4, 5, 6, 7\}, \{(56, 2), (23, 7)\})$.

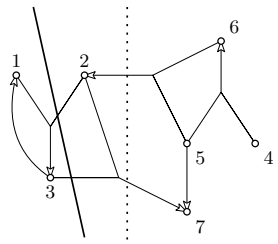


Fig. 1: The full line illustrates a bipartition which is not a split, whereas the dotted line corresponds to a split.

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 edge. Then any bipartition of the ground set 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 edge. 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 edges 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. Figure 2 shows a body-connected dihypergraph.

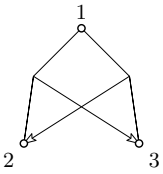


Fig. 2: A body-connected dihypergraph

First observe that a body reduced to a singleton always satisfies condition of Definition 1. Thus, unit edges of a dihypergraph have no impact on a split. Next, we give a characterization of dihypergraphs that have 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 edge $e_i = (B_i, h_i)$ cannot satisfy the condition 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 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 maximal 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 **??** with the split $\mathrm{V}_1 = \{1, 2, 3\}$ and $\mathrm{V}_2 = \{4, 5, 6, 7\}$. Then 5 and 6 was body-connected in $\mathcal{H}$ but not in $\mathcal{H}[\mathrm{V}_2]$. Therefore, body-connected components may be decomposed in turn. 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. 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 the same, the order of the children of an interior node is not important.

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:*

1. *$\lambda(v) \in \mathrm{V}$ if v is a leaf of T,*
2. *$\lambda(v) \subseteq \mathcal{E}$ if v is an interior node (possibly $\lambda(v) = \emptyset$),*
3. *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.*
4. *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 3 shows a $\mathcal{H}$-tree for the dihypergraph in Figure 1.

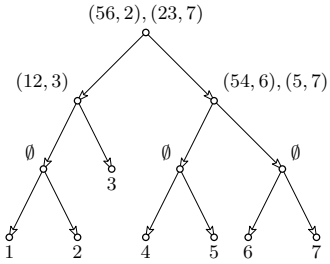


Fig. 3: A H-decomposition for the dihypergraph in Figure 1

There are two interesting cases where a H-decomposition of a dihypergraph $\mathcal{H}$ can be computed easily (see Figure 4).

- the dihypergraph $\mathcal{H}$ has no edges. 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*. The same as for the previous case, except that an edge (b, h) will be in the label of the least common ancestor of the leaves labelled by b and h.

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

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.



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

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 $\mathcal{E}(\mathcal{H}[U])$.

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 a $\mathcal{H}[C]$ 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 edge 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 edge 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 edge 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 $\mathcal{H}[\mathrm{V} \setminus C]$. Therefore, condition *(iii)* is satisfied and (T, λ) is a $\mathcal{H}$-tree, concluding the proof.

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

then we recursively call the algorithm on the subhypergraphs $\mathcal{H}[C]$ and $\mathcal{H}[\mathrm{V}\setminus C]$. This strategy is formalized in Algorithm 1, whose correctness and complexity are studied in Theorem 2.

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 ;

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. We first show using induction on the set of vertices $|\mathrm{V}|$ that the algorithm returns a $\mathcal{H}$-tree iff $\mathcal{H}$ is H-decomposable. Clearly, for dihypergraphs containing only one vertex x, the algorithm returns a $\mathcal{H}$-tree corresponding to a leaf with label x. Now, assume that the algorithm is correct for dihypergraphs with $|\mathrm{V}| < n, \in \mathbb{N}$, and consider a dihypergraph $\mathcal{H}$ with $|\mathrm{V}| = n$.

Suppose $\mathcal{H}$ is H-decomposable. Then $\mathcal{H}$ is not body-connected by Proposition 1. Let C be a body-connected component of $\mathcal{H}$. Inductively, the algorithm is correct for $\mathcal{H}[C]$ and $\mathcal{H}[\mathrm{V}\setminus C]$ since $1 \leq |C| < n$. So by Theorem 1, $\mathcal{H}[C]$ 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). Therefore, the algorithm returns a tree 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 which satisfies all conditions for (T, λ) to be a $\mathcal{H}$-tree. Thus the algorithm computes a $\mathcal{H}$-tree for all dihypergraphs that are H-decomposable.

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

1. If $\mathcal{H}$ is body-connected then the algorithm returns FAIL in Line 7.
2. If $\mathcal{H}$ 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.

Therefore, the algorithm fails whenever 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 structures in [TR84], which runs in almost linear time, i.e., $O(|\,\mathcal{H}\,| \cdot \alpha(|\,\mathcal{H}\,|, |\,\mathrm{V}\,|))$ where $\alpha(.,.)$ is the inverse of the Ackermann function. The almost linear comes from the fact that $\alpha(|\,\mathrm{V}\,|) \leq 4$ for any practical dihypergraph. 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 the a body-connected component in line **5**. Thus, there are many possible $\mathcal{H}$-trees that represent a hierarchical decomposition of a given dihypergraph. Then, a natural question arises: are all *$\mathcal{H}$-trees equivalently interesting?* Figure 5 shows two possible $\mathcal{H}$-trees for the dihypergraph $\mathcal{H} = (\mathrm{V}, \mathcal{E})$ with $\mathrm{V} = [8]$ and $\mathcal{E} = \{(12,3), (23,4), (34,5), (56,7), (67,8)\}$.

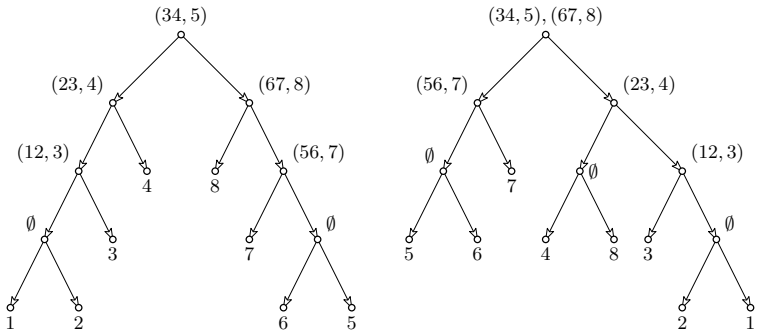


Fig. 5: Two possible $\mathcal{H}$-trees of the same dihypergraph

3.3 Extension of the H-decomposition

A 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* $\lambda(r) = \mathcal{E}$*;*
return r;

Figure 6 illustrates the H-decomposition of a dihypergraph, where the left most leaf corresponds to a H-factor which is not trivial.

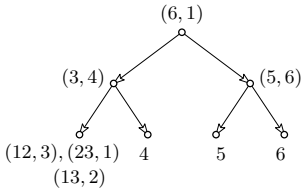


Fig. 6: H-decomposition into H-factors

Now, any dihypergraph has a H-decomposition into H-factors, and then it 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

Decompositions of closure systems or lattices has been widely studied either from the lattice itself [Grä11, GW12], from a context [GW99, VBDM15b] or from the database aspect [Lib93, DLM92].

Decomposition of closure systems is of interest for many applications in Formal Concept Analysis ([GW12, VBDM15b, KVD05]) such as social networks and datamining. Closure systems are usualy represented by a binary matrix, also known as context [GW99, Wil17]. In this section, we consider closure systems represented by dihypergraphs, see [Wil17, AL17], and show that the H-decomposition introduced in the previous section can be applied to closure system decomposition.

We first recall some definitions for closure systems and lattice theory. The reader can refer to [Grä11] for a thorough introduction to the topic. 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 $y \leq x$, and *incomparable otherwise*. An upper bound of x, y is an element $u \in \mathcal{L}$ such that $x \leq u$, $y \leq u$. If for any upper bound $u' \neq u$, $u \leq u'$, then u is the *least upper bound* of x, y, written $x \vee y$. Lower bounds and the greatest lower bound $x \wedge y$ are defined dually. We say that $\mathcal{L}$ is a *lattice* if for any $x, y \in \mathcal{L}$, $x \vee y$ and $x \wedge y$ are well defined. A *meet-sublattice* $\mathcal{L}'$ of $\mathcal{L}$ is a subset of elements of $\mathcal{L}$ such that for any $x, y \in \mathcal{L}'$ $x \wedge y \in \mathcal{L}'$. A meet-sublattice $\mathcal{L}'$ of $\mathcal{L}$ is a *sublattice* of $\mathcal{L}$ if $x \vee y \in \mathcal{L}'$. Among elements of $\mathcal{L}$, we say that x is a *join-irreducible* if for any $y, z \in \mathcal{L}$, $x = y \vee z$ implies $x = y$ or $x = z$. The set of join-irreducible elements of $\mathcal{L}$ is denoted by $\mathrm{J}(\mathcal{L})$.

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 with partial ordering by set containment is always a lattice. Dually, to any lattice $\mathcal{L}$ is associated to a closure system on its join-irreducible elements. The lattice $\mathcal{L}$ is isomorphic to the closure system $\{J_x \mid x \in \mathcal{L}\}$ when ordered by set containment, where $J_x = \{j \in \mathrm{J}(\mathcal{L}) \mid j \leq x\}$.

The *projection* of a closure system $\mathcal{F}$ over a subset $U \subseteq \mathrm{V}$, named here *trace* and noted $\mathcal{F}\colon U$, 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 trace $\mathcal{F}\colon U$ is always a sublattice of the lattice $(\mathcal{F}, \subseteq)$.

First, we recall the *forward chaining* method for computing the closure system from its associated dihypergraph. Let $\mathcal{H}$ be a dihypergraph and $X \subseteq \mathrm{V}$, we construct a chain of subsets of V $X = X_0 \subset X_1 \subset \ldots \subset X_k = X^{\mathcal{H}}$, where $X_i = X_{i-1} \cup \{h | B \subseteq X_{i-1}, (B, h) \in \mathcal{E}\}$ with $i > 0$. The subset $X^{\mathcal{H}}$ is called a fixed point or a closed set. Indeed, a subset F of V is closed, if for any edge $(B, h), B \subseteq F$ implies $h \in F$. The set of all closed sets $\mathcal{F}_{\mathcal{H}} = \{X^{\mathcal{H}} \mid X \subseteq \mathrm{V}\}$ is a closure system. Notice, that there are many dihypergraphs that lead to the same closure system.

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 the closure system $\mathcal{F}_{\mathcal{H}}$ is obtained from the H-decomposition of the dihypergraph $\mathcal{H}$, 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 7 illustrates the H-decomposition of the closure system associated to the H-decomposition of the dihypergraph in Figure 6.

Next, we study properties of the three closure systems corresponding to the three subhypergraphs induced by a split of the dihypergraph.

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 edge then $\mathcal{F}_{\mathcal{H}} = \mathcal{F}_1 \times \mathcal{F}_2$.*
3. *If every edge (B, h) of $\mathcal{H}[\mathrm{V}_1, \mathrm{V}_2]$, we have $B \subseteq \mathrm{V}_1$ then $\mathcal{F}_{\mathcal{H}} : \mathrm{V}_1 = \mathcal{F}_1$ and $\mathcal{F}_{\mathcal{H}} : \mathrm{V}_2 = \mathcal{F}_2$.*
4. *If every edge (B, h) of $\mathcal{H}[\mathrm{V}_1, \mathrm{V}_2]$, we have $B \subseteq \mathrm{V}_2$ then $\mathcal{F}_{\mathcal{H}} : \mathrm{V}_2 = \mathcal{F}_2$ and $\mathcal{F}_{\mathcal{H}} : \mathrm{V}_1 = \mathcal{F}_1$.*

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 proof (i), (iii) and (ii). The item (iv) is similar to (iii).

(i) Let $F \in \mathcal{F}_{\mathcal{H}}$ with $F_i = F \cap \mathrm{V}_i$ and (B, h) an edge of $\mathcal{H}[\mathrm{V}_i]$ for some $i \in \{1, 2\}$. Suppose $B \subseteq F_i$ and $h \notin F_i$. Then we also have $B \subseteq F$ and $h \notin F$ which contradicts that $F \in \mathcal{F}_{\mathcal{H}}$, since (B, h) is an edge of $\mathcal{H}$.
(iii) Without loss of generality, we prove the case for $i = 1$. Let $F \in \mathcal{F}_1$, we show that $F^{\mathcal{H}}$ (the forward chaining applied to F in $\mathcal{H}$) satisfies $F^{\mathcal{H}} \cap \mathrm{V}_1 = F$. Let (B, h) an edge of $\mathcal{H}$. We distinguish 3 cases:

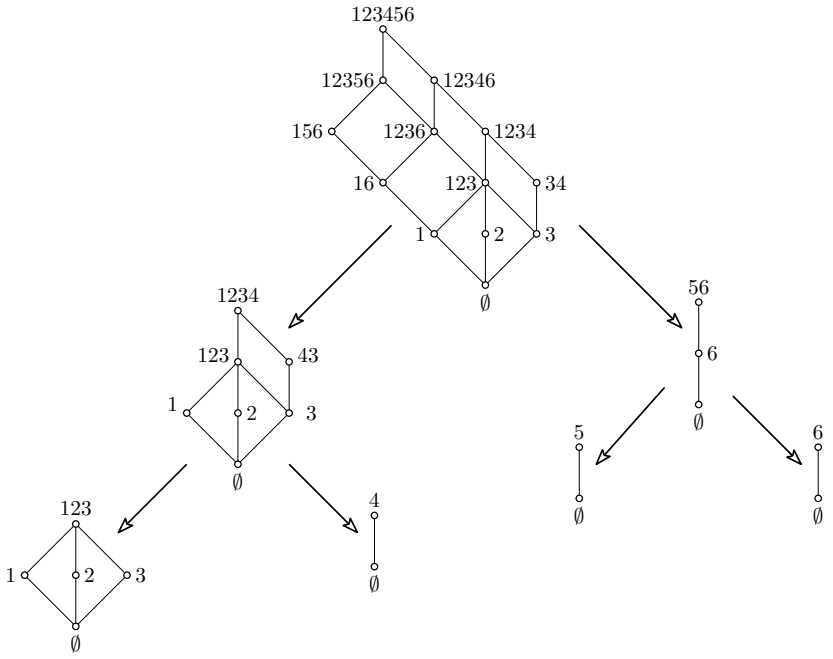


Fig. 7: The H-decomposition of the closure system corresponding to the dihypergraph in Figure 6

(a) if $B \subseteq \mathrm{V}_2$ then $B \not\subseteq F$. Thus the edge (B, h) has no effect in the forward chaining.
(b) if $B \subseteq \mathrm{V}_1$ and $h \in \mathrm{V}_1$ then $B \subseteq F$ implies F contains h since it is closed in $\mathcal{H}[\mathrm{V}_1]$.
(c) if $B \subseteq \mathrm{V}_1$ and $h \in \mathrm{V}_2$ then (B, h) is an edge of $\mathcal{H}[\mathrm{V}_1, \mathrm{V}_2]$. Then there is no edge outside $\mathcal{H}[\mathrm{V}_1]$ with head in V_1, and thus the forward chaining cannot add an element from V_1.

So $F \in$ is in the trace of $\mathcal{F}_{\mathcal{H}}$ over V_1.
The converse is true by the proof of (i).
(ii) Since $\mathcal{H}$ has no edge, then it satisfy (iii). We deduce that $\mathcal{F}_{\mathcal{H}} : \mathrm{V}_1 = \mathcal{F}_1$ and $\mathcal{F}_{\mathcal{H}} : \mathrm{V}_2 = \mathcal{F}_2$. 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 edge of $\mathcal{H}$ such that $B \subseteq F_1 \cup F_2$. Since $\mathcal{H}[\mathrm{V}_1, \mathrm{V}_2]$ has no edge, then (B, h) is an edge of $\mathcal{H}[\mathrm{V}_1]$ or an edge of $\mathcal{H}[\mathrm{V}_2]$. In any case F_1 or F_2 contains h. We conclude that $F_1 \cup F_2 \in \mathcal{F}_{\mathcal{H}}$.

According to Theorem 3 (i), any closure closure 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 objective was mixed, since the size of $\mathcal{F}_1$ and $\mathcal{F}_2$ may be exponential in the size of $\mathcal{F}_{\mathcal{H}}$. This is the case, when the subhypergraphs $\mathcal{H}[\mathrm{V}_1]$ and $\mathcal{H}[\mathrm{V}_2]$ have no

edge, and the edges of $\mathcal{H}[\mathrm{V}_1, \mathrm{V}_2]$ are as follows: $\mathcal{E}_1 \cup \mathcal{E}_2 = \{(vv', u) \in {\mathrm{V}_1}^2 \times U_2 \mid v \neq v'\} \cup \{(uu', v) \in {\mathrm{V}_2}^2 \times \mathrm{V}_1 \mid v \neq v'\}$. Then, $\mathcal{F}_1 = 2^{\mathrm{V}_1}$ and $\mathcal{F}_2 = 2^{\mathrm{V}_2}$ which are exponential sizes, whereas the closure system $\mathcal{F}_{\mathcal{H}}$ has $|\mathrm{V}_1| \times |\mathrm{V}_2| + |\mathrm{V}| + 2$ elements, namely $\emptyset$, V, any singleton element $v \in \mathrm{V}$, and any pair $vu \in \mathrm{V}_1 \times \mathrm{V}_2$. It is worth noticing that this combinatorial explosion cannot happen whenever $\mathcal{F}_1$ and $\mathcal{F}_2$ are traces. In this case, the size of the closure system $\mathcal{F}_{\mathcal{H}}$ is most twice the maximum size of $\mathcal{F}_1$ and $\mathcal{F}_2$.

However, the H-decomposition allows us to go further in the decomposition of closure systems and lattices approaching the most famous Birkhoff's theorem of the subdirect decomposition which states "*Every algebra is a subdirect product of its subdirectly irreducible homomorphic*". The interpretation in our approach is that "Every closure system is a sublattice of the direct product of irreducible traces". Irreducible traces are closure systems that cannot be obtained as a sublattice of the direct product of its traces. Consider the closure system $\mathcal{F}_{\mathcal{H}}$ in Figure 8(a) encoded by the unique dihypergraph $\mathcal{H} = (\{1, 2, 3\}, \{(2, 1), (13, 2)\})$. It is known that it can not be obtained as a sublattice of the direct product of traces. Clearly $\mathcal{H}$ is not body-connected and thus $\mathrm{V}_1 = \{1, 3\}$ 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 $\mathcal{F}_{\mathcal{H}}$.

Figure 8(b), (c) and (d) are subdirectly irreducible and H-factors too.

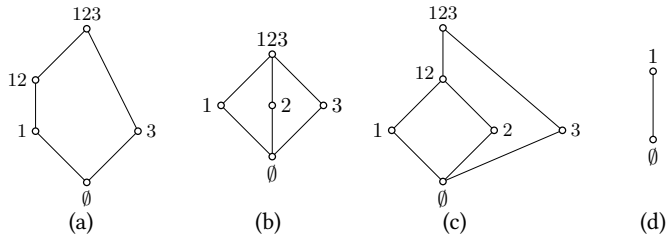


Fig. 8

We conclude this 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.

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.

[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.

[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.

[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.