Path-Based Supports for Hypergraphs

Ulrik Brandes, Sabine Cornelsen, Barbara Pampel, Arnaud Sallaberry

Path-Based Supports for Hypergraphs

Ulrik Brandes[1], Sabine Cornelsen[1], Barbara Pampel[1], and Arnaud Sallaberry[2]

[1] Fachbereich Informatik & Informationswissenschaft, Universität Konstanz
{Ulrik.Brandes,Sabine.Cornelsen,Barbara.Pampel}@uni-konstanz.de
[2] CNRS UMR 5800 LaBRI, INRIA Bordeaux - Sud Ouest, Pikko
arnaud.sallaberry@labri.fr

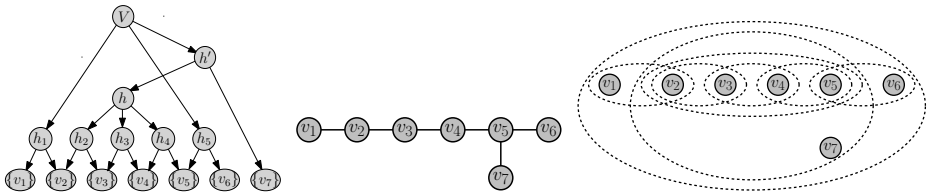
Abstract. A path-based support of a hypergraph H is a graph with the same vertex set as H in which each hyperedge induces a Hamiltonian subgraph. While it is $\mathcal{NP}$-complete to compute a path-based support with the minimum number of edges or to decide whether there is a planar path-based support, we show that a path-based tree support can be computed in polynomial time if it exists.

1 Introduction

A *hypergraph* is a pair $H = (V, A)$ where V is a finite set and A is a (multi-)set of non-empty subsets of V. The elements of V are called *vertices* and the elements of A are called *hyperedges*. A *support* (or *host graph*) of a hypergraph $H = (V, A)$ is a graph $G = (V, E)$ such that each hyperedge of H induces a connected subgraph of G, i.e., such that the graph $G[h] := (h, \{e \in E, e \subseteq h\})$ is connected for every $h \in A$. See Fig. 1(b) for an example.

Applications for supports of hypergraphs are, e.g., in hypergraph coloring [10, 4], databases [1], or hypergraph drawing [7, 8, 3, 12]. E.g., see Fig. 1 for an application of a support for designing Euler diagrams. An *Euler diagram* of a hypergraph $H = (V, A)$ is a drawing of H in the plane in which the vertices are drawn as points and each hyperedge $h \in A$ is drawn as a simple closed region containing the points representing the vertices in h and not the points representing the vertices in $V \setminus h$. There are various well-formedness conditions for Euler diagrams, see e.g. [5, 12].

Recently the problem of deciding which classes of hypergraphs admit what kind of supports became of interest again. It can be tested in linear time whether a hypergraph has a support that is a tree [13], a path or a cycle [3]. It can be decided in polynomial time whether a hypergraph has a tree support with bounded degrees [3] or a cactus support [2]. A minimum weighted tree support can be computed in polynomial time [9]. It is $\mathcal{NP}$-complete to decide whether a hypergraph has a planar support [7], a compact support [7, 8] or a 2-outerplanar support [3]. A support with the minimum number of edges can be computed in polynomial time if the hypergraph is closed under intersections [3]. If the set of hyperedges is closed under intersections and differences, it can be decided in polynomial time whether the hypergraph has an outerplanar support [2].



(a) Hasse diagram (b) tree support (c) Euler diagram

Fig. 1. Three representations of the hypergraph $H = (V, A)$ with hyperedges $h_1 = \{v_1, v_2\}$, $h_2 = \{v_2, v_3\}$, $h_3 = \{v_3, v_4\}$, $h_4 = \{v_4, v_5\}$, $h_5 = \{v_5, v_6\}$, $h = \{v_2, v_3, v_4, v_5\}$, $h' = \{v_2, v_3, v_4, v_5, v_7\}$, and $V = \{v_1, \ldots, v_7\}$.

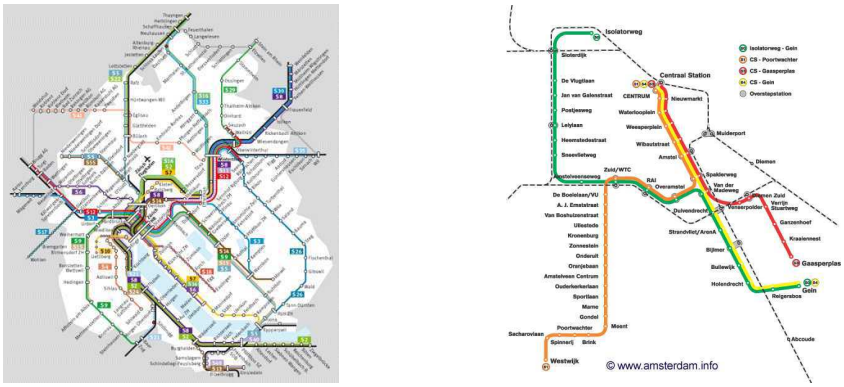
In this paper, we consider a restriction on the subgraphs of a support that are induced by the hyperedges. A support G of a hypergraph $H = (V, A)$ is called *path-based* if the subgraph $G[h]$ contains a *Hamiltonian path* for each hyperedge $h \in A$, i.e., $G[h]$ contains a path that contains each vertex of h. This approach was on one hand motivated by hypergraph drawing and on the other hand by the aesthetics of metro map layouts. I.e., the hyperedges could be visualized as lines along the Hamiltonian path in the induced subgraph of the support like the metro lines in a metro map. See Fig. 2 for examples of metro maps and Fig. 3(c) for a representation of some hyperedges in such a metro map like drawing. For metro map layout algorithms see, e.g., [11, 14].

We briefly consider planar path-based supports and minimum path-based supports. Our main result is a characterization of those hypergraphs that have a path-based tree support and a polynomial time algorithm for constructing path-based tree supports if they exist. E.g., Fig. 1 shows an example of a hypergraph $H = (V, A)$ that has a tree support but no path-based tree support. However, the tree support in Fig. 1(b) is a path-based tree support for $(V, A \setminus \{V\})$.

The contribution of this paper is as follows. In Section 2, we give the necessary definitions. We then briefly mention in Section 3 that finding a minimum path-based support or deciding whether there is a planar path-based support, respectively, is $\mathcal{NP}$-complete. We consider path-based tree supports in Sect. 4. In Section 4.1, we review a method for computing tree supports using the Hasse diagram. In Section 4.2, we show how to apply this method to test whether a hypergraph has a path-based tree support and if so how to compute one in polynomial time. Finally, in Section 4.3 we discuss the run time of our method.

2 Preliminaries

In this section, we give the necessary definitions that were not already given in the introduction. Throughout this paper let $H = (V, A)$ be a hypergraph. We denote by $n = |V|$ the number of vertices, $m = |A|$ the number of hyperedges, and $N = \sum_{h \in A} |h|$ the sum of the sizes of all hyperedges of a hypergraph H. The *size of the hypergraph* H is then $N + n + m$. A hypergraph is a *graph* if



(a) local trains of Zurich (b) metro of Amsterdam

Fig. 2. Local train map of Zurich (`www.zvv.ch`) and the metro map of Amsterdam (`www.amsterdam.info`). In (b) the union of all lines forms a tree.

all hyperedges contain exactly two vertices. A hypergraph $H = (V, A)$ is *closed under intersections* if $h_1 \cap h_2 \in A \cup \{\emptyset\}$ for $h_1, h_2 \in A$.

The *Hasse diagram* of a hypergraph $H = (V, A)$ is the directed acyclic graph with vertex set $A \cup \{\{v\}; v \in V\}$ and there is an edge (h_1, h_2) if and only if $h_2 \subsetneq h_1$ and there is no set $h \in A$ with $h_2 \subsetneq h \subsetneq h_1$. Fig. 1(a) shows an example of a Hasse diagram. Let (v, w) be an edge of a directed acyclic graph. Then we say that w is a *child* of v and v a *parent* of w. For a *descendant* d of v there is a directed path from v to d while for an *ancestor* a of v there is a directed path from a to v. A *source* does not have any parents, a *sink* no children and an *inner vertex* has at least one parent and one child.

3 Minimum and Planar Path-Based Supports

Assuming that each hyperedge contains at least one vertex, each hypergraph $H = (V, A)$ has a path-based support $G = (V, E)$ with at most $N - m$ edges: Order the vertices arbitrarily. For each hyperedge $\{v_1, \ldots, v_k\} \in A$ with $v_1 < \cdots < v_k$ with respect to that ordering the edge set E contains $\{v_{i-1}, v_i\}, i = 1, \ldots, k$. It is, however, $\mathcal{NP}$-complete to find an ordering of the vertices that minimizes the number of edges of the thus constructed path-based support of H [6]. Moreover, even if we had an ordering of the vertices that had minimized the number of the thus constructed path-based support, this support still does not have to yield the minimum number of edges in any path-based support of H. E.g., consider the hypergraph with hyperedges $\{1, 2, 4\}$, $\{1, 3, 4\}$, and $\{2, 3, 4\}$. Nevertheless, we have the following theorem.

Theorem 1. *It is $\mathcal{NP}$-complete to minimize the number of edges in a path-based support of a hypergraph – even if it is closed under intersections.*

Proof. Reduction from Hamiltonian path. Let $G = (V, E)$ be a graph. Let $H = (V, E \cup \{V\} \cup \{\{v\};\ v \in V\})$ and $K = |E|$. Then G contains a Hamiltonian path if and only if H has a path-based support with at most K edges. □

For the application of Euler diagram like drawings, planar supports are of special interests. However, like for general planar supports, the problem of testing whether there is a path-based planar support is hard.

Theorem 2. *It is $\mathcal{NP}$-complete to decide whether a hypergraph – even if it is closed under intersections – has a path-based planar support.*

Proof. The support that Johnson and Pollak [7] constructed to prove that it is $\mathcal{NP}$-complete to decide whether there is a planar support was already path-based. □

4 Path-Based Tree Supports

In this section we show how to decide in polynomial time whether a given hypergraph has a path-based tree support. If such a support exists, it is at the same time a path-based support of minimum size and a planar path-based support. So far it is known how to decide in linear time whether there is a path-based tree support if $V \in A$ [3].

4.1 Constructing a Tree Support from the Hasse Diagram

A support with the minimum number of edges and, hence, a tree support if one exists can easily be constructed from the Hasse diagram if the hypergraph is closed under intersections [3].

To construct a tree support of an arbitrary hypergraph, it suffices to consider the *augmented Hasse diagram* – a representation of "necessary" intersections of hyperedges. The definition is as follows. First consider the smallest set $\overline{A}$ of subsets of V that contains A and that is closed under intersections. Consider the Hasse diagram $\overline{D}$ of $\overline{H} = (V, \overline{A})$. Note that any tree support of H is also a tree support of $\overline{H}$. Let $h_1, \ldots, h_k$ be the children of a hyperedge h in $\overline{D}$. The hyperedge $h \in \overline{A}$ is *implied* if the hypergraph $(h_1 \cup \cdots \cup h_k, \{h_1, \ldots, h_k\})$ is connected and *non-implied* otherwise. Let $\{h_1, \ldots, h_k\}$ be a maximal subset of the children of a non-implied hyperedge in $\overline{A}$ such that $(h_1 \cup \cdots \cup h_k, \{h_1, \ldots, h_k\})$ is connected. Then $h_1 \cup \cdots \cup h_k$ is a *summary hyperedge*. Note that a summary hyperedge does not have to be in $\overline{A}$. Let A' be the set of subsets of V containing the summary hyperedges, the hyperedges in $\overline{A}$ that are not implied, and the sources of $\overline{D}$. E.g., for the hypergraph in Fig. 1 it holds that $A' = A$. In this example, the hyperedge h is a summary hyperedge, h' is not implied, and V is a source.

The augmented Hasse diagram of H is the Hasse diagram D' of $H' = (V, A')$. If H has a tree support then the augmented Hasse diagram has $\mathcal{O}(n+m)$ vertices

and can be constructed in $\mathcal{O}(n^3 m)$ time [3]. Further note that if H has a tree support and $h \in A'$ is non-implied then all children of h in D' are disjoint.

If a tree support $G = (V, E)$ of H exists it can be constructed as follows [3]. Starting with an empty graph G, we proceed from the sinks to the sources of D'. If $h \in A'$ is not implied, choose an arbitrary ordering $h_1, \ldots, h_k$ of the children of h in D'. We assume that at this stage, $G[h_i], i = 1, \ldots, k$ are already connected subgraphs of G. For $j = 2, \ldots, k$, choose vertices $v_j \in \bigcup_{i=1}^{j-1} h_i$, $w_j \in h_j$ and add edges $\{v_j, w_j\}$ to E.

If we want to construct a path-based tree support, then $G[h_j], j = 1, \ldots, k$ are paths and as vertices v_{j+1} and w_j for the edges connecting $G[h_j]$ to the other paths, we choose the end vertices of $G[h_j]$. The only choices that remain is the ordering of the children of h and the choice of which end vertex of $G[h_j]$ is w_j and which one is v_{j+1}. The implied hyperedges give restrictions on how these choices might be done.

4.2 Choosing the Connections: A Characterization

When we want to apply the general method introduced in Sect. 4.1 to construct a path-based tree support G, we have to make sure that we do not create vertices of degree greater than 2 in $G[h]$ when processing non-implied hyperedges contained in an implied hyperedge h.

Let $h', h'' \in A'$. We say that h', h'' overlap if $h' \cap h'' \neq \emptyset$, $h' \not\subseteq h''$, and $h'' \not\subseteq h'$. Two overlapping hyperedges $h', h'' \in A'$ have a *conflict* if there is some hyperedge in A' that contains h' and h''. Two overlapping hyperedges $h', h'' \in A'$ have a *conflict with respect to* $h \in A'$ if h' has a conflict with h'', $h' \cap h'' \subseteq h$ and h is a child of h' or h''. In that case we say that h' and h'' are *conflicting* hyperedges of h. Let A'_h be the set of conflicting hyperedges of h. Let A^c_h be the set of children h_i of h such that $h \in A'_{h_i}$.

Assume now that H has a path-based tree support G and let $h, h', h'' \in A'$ be such that h' and h'' have a conflict with respect to h. We have three types of restrictions on the connections of the paths.

1. $G[h' \setminus h]$ and $G[h'' \setminus h]$ are paths that are attached to different end vertices of $G[h]$. Otherwise $G[h_a]$ contains a vertex of degree higher than 2 for any hyperedge $h_a \supseteq h' \cup h''$.
2. Assume further that $h_1 \in A^c_h$. For all hyperedges $h^1 \in A'_h$ that have a conflict with h with respect to h_1 it holds that $G[h^1 \setminus h]$ has to be appended to the end vertex of $G[h]$ that is also an end vertex of $G[h_1]$. Hence, all these paths $G[h^1 \setminus h]$ have to be appended to the same end vertex of $G[h]$.
3. Assume further that $h_2 \in A^c_h, h_2 \neq h_1$. Let $h^i \in A'_h$ have a conflict with h with respect to $h_i, i = 1, 2$, respectively. Then $G[h^i \setminus h]$ has to be appended to the end vertex of $G[h]$ that is also an end vertex of $G[h_i]$. Hence, $G[h^1 \setminus h]$ and $G[h^2 \setminus h]$ have to be appended to different end vertices of $G[h]$.

E.g., consider the hypergraph $H = (V, A)$ in Fig. 1. Then on one hand, h' has a conflict with h_1 and h_5 with respect to h. Hence, by the first type of restrictions

$G[h_1 \setminus h]$ and $G[h_5 \setminus h]$ have to be appended to the same end vertex of $G[h]$, i.e. the end vertex of $G[h]$ to which $G[h' \setminus h]$ is not appended. On the other hand, h_1 and h have a conflict with respect to h_2 while h_5 and h have a conflict with respect to h_4. Hence, by the third type of restrictions it follows that $G[h_1 \setminus h]$ and $G[h_5 \setminus h]$ have to be appended to different end vertices of $G[h]$. Hence, there is no path-based tree support for H.

This motivates the following definition of conflict graphs. The *conflict graph* $C_h, h \in A'$ is a graph on the vertex set $A'_h \cup A^c_h$. The conflict graph C_h contains the following three types of edges.

1. $\{h', h''\}, h', h'' \in A'_h$ if h' and h'' have a conflict with respect to h.
2. $\{h', h_1\}, h' \in A'_h, h_1 \in A^c_h$ if $h' \in A'_{h_1}$ and h' and h have a conflict with respect to h_1.
3. $\{h_1, h_2\}, h_1, h_2 \in A^c_h, h_1 \neq h_2$.

E.g., consider the hypergraph $H = (V, A)$ in Fig. 1. Then the conflict graph C_h contains the edges $\{h', h_5\}$ and $\{h', h_1\}$ of type one, the edges $\{h_2, h_1\}$ and $\{h_4, h_5\}$ of type 2 and the edge $\{h_2, h_4\}$ of type 3. Hence, C_h contains a cycle of odd length, reflecting that there is no suitable assignment of the end vertices of $G[h]$ to h_1, h_5 and h'.

Theorem 3. *A hypergraph $H = (V, A)$ has a path-based tree support if and only if*

1. *H has a tree support,*
2. *no hyperedge contains three pairwise overlapping hyperedges $h_1, h_2, h_3 \in A'$ with $h_1 \cap h_2 = h_2 \cap h_3 = h_1 \cap h_3$, and*
3. *all conflict graphs $C_h, h \in A', |h| > 1$ are bipartite.*

From the observations before the definition of the conflict graph it is clear that the conditions of Theorem 3 are necessary for a path-based tree support. In the remainder of this section, we prove that the conditions are also sufficient.

In the following assume that the conditions of Theorem 3 are fulfilled. We show in Algorithm 1 how to construct a path-based tree support G of H. We consider the vertices of the augmented Hasse diagram D' from the sinks to the sources in a *reversed topological order*, i.e., we consider a hyperedge only if all its children in D' have already been considered. During the algorithm, a conflicting hyperedge h' of a hyperedge h is labeled with the end vertex v of $G[h]$ if the path $G[h' \setminus h]$ will be appended to v. We will call this label $\text{side}_h(h')$. Concerning Step 2a, the sets $A^c_h, h \in A'$ contain at most two hyperedges – otherwise the subgraph of C_h induced by A^c_h contains a triangle and, hence, is not bipartite.

Algorithm 1 constructs a tree support G of H [3]. Before we show that G is a path-based tree support, we illustrate the algorithm with an example. Consider the hypergraph H in Fig. 3. We show how the algorithm proceeds h^5_1 and all its descendants in D'. For the hyperedges h^1_3, h^1_4, h^1_6, and h^1_8 the conflict graphs are empty while for the other leaves we have $\text{side}_{h^1_5}(h^2_2) = \text{side}_{h^1_5}(h^2_3) = \text{side}_{h^1_5}(h^3_1) = \text{side}_{h^1_5}(h^4_2) = v_5$, $\text{side}_{h^1_7}(h^2_4) = \text{side}_{h^1_7}(h^3_1) = v_7$, and $\text{side}_{h^1_9}(h^2_4) =$

$\text{side}_{h_9^1}(h_1^4) = \text{side}_{h_9^1}(h_5^2) = \text{side}_{h_9^1}(h_6^2) = \text{side}_{h_9^1}(h_7^2) = v_9$. When operating h_2^2 and h_3^2, respectively, we add edges $\{v_4, v_5\}$ and $\{v_5, v_6\}$, respectively, to G. While the conflict graph of h_2^2 does only contain h_5^1 with $\text{side}_{h_2^2}(h_5^1) = v_4$, in $C_{h_3^2}$ we set $\text{side}_{h_3^2}(h_5^1) = \text{side}_{h_3^2}(h_1^3) = v_6$, and $\text{side}_{h_3^2}(h_2^2) = v_5$. h_4^2 has a conflict with respect to h_7^1 and h_9^1. Hence, we add edges $\{v_7, v_8\}$ and $\{v_8, v_9\}$ to G. Further, $\text{side}_{h_4^2}(h_7^1) = \text{side}_{h_4^2}(h_5^2) = v_9$ and $\text{side}_{h_4^2}(h_9^1) = \text{side}_{h_4^2}(h_1^4) = v_7$. When operating h_1^3 we can choose $h_1 = h_3^2$ and $h_2 = h_7^1$, since $\text{side}_{h_3^2}(h_3^1) = v_6$ and $\text{side}_{h_7^1}(h_3^1) = v_7$. We add the edge $\{v_6, v_7\}$ to G. The conflict graph $C_{h_1^3}$ is shown in Fig. 3(b). The hyperedge h_1^4 is implied and we set $\text{side}_{h_1^4}(h_4^2) = v_4$. We can finally connect v_3 to v_4 or v_9 when operating h_1^5.

To prove the correctness of Algorithm 1, it remains to show that all hyperedges of H induce a path in G. Since we included all inclusion maximal hyperedges of H in A', it suffices to show this property for all hyperedges in A'. We start with a technical lemma.

Lemma 1. *Let h' and h'' be two overlapping hyperedges and let h' be not implied. Then there is a hyperedge $h \in A'$ with $h' \cap h'' \subseteq h \subsetneq h'$.*

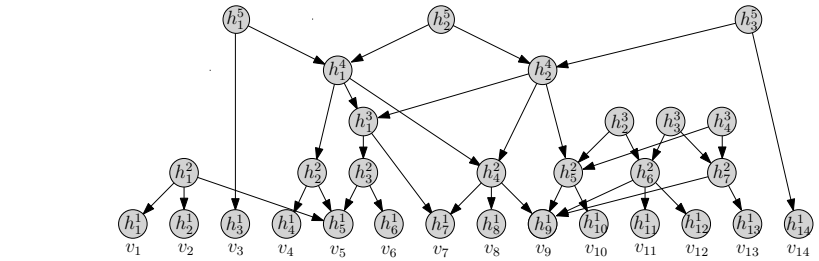
Proof. Let $h_c \in \overline{A}$ be maximal with $h' \cap h'' \subseteq h_c \subsetneq h'$. The hyperedge h_c is a child of the non-implied hyperedge h' in $\overline{D}$. Consider the summary hyperedge h with $h_c \subseteq h \subsetneq h'$. By definition of A' it follows that $h \in A'$. □

For an edge $\{v, w\}$ of G let h_{vw} be the intersection of all hyperedges of A' that contain v and w. Note that then h_{vw} is not implied since v and w cannot both be contained in a subset of h_{vw}. Hence, $h_{vw} \in A'$.

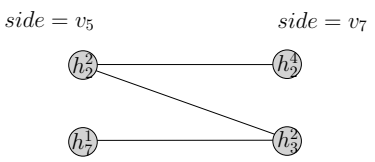
Algorithm 1: Path-based tree support

Let $E = \emptyset$.
For $h \in A'$ in a reversed topological order of D'.

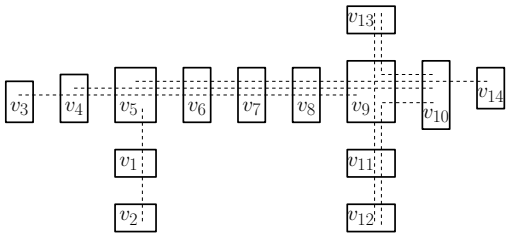
1. If $h = \{v\}$ for some $v \in V$
   (a) set $\text{side}_h(h') = v$ for all vertices h' of C_h.
2. Else
   (a) let $h_1, \ldots, h_k$ be the children of h such that $h_2, \ldots, h_{k-1} \notin A_h^c$.
   (b) If h is non-implied
      i. let $w_i, v_{i+1}, i = 1, \ldots, k$ be the end vertices of $G[h_i]$ such that
         A. $\text{side}_{h_1}(h) = v_2$ if $h \in A'_{h_1}$ and
         B. $\text{side}_{h_k}(h) = w_k$ if $h \in A'_{h_k}$.
      ii. Add the edges $\{v_i, w_i\}, i = 2, \ldots, k$ to E.
   (c) Else let $w_1 \neq v_{k+1}$ be the end vertices of $G[h]$ such that
      i. $v_{k+1} \notin h_1$ and
      ii. $w_1 \notin h_k$.
   (d) If $h_1 \in A_h^c$ set $\text{side}_h(h_1) = v_{k+1}$.
   (e) If $h_k \in A_h^c$ set $\text{side}_h(h_k) = w_1$.
   (f) Label the remaining vertices of C_h with v_{k+1} or w_1 such that no two adjacent vertices have the same label.



(a) augmented Hasse diagram D' of a hypergraph H



(b) conflict graph of hyperedge h_1^3



(c) metro map like drawing of the sources of D'

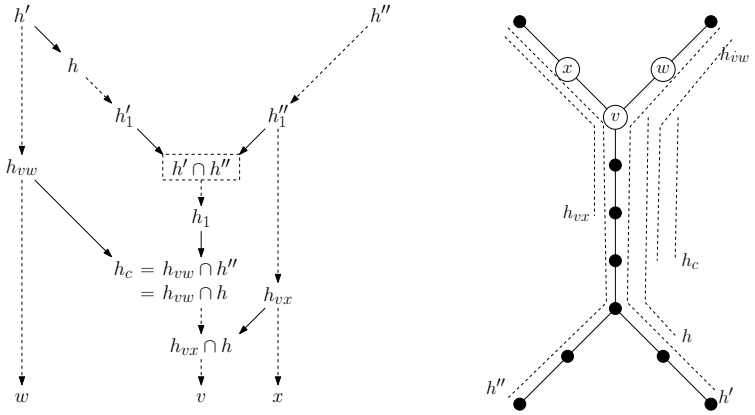
Fig. 3. Illustration of Algorithm 1.

Lemma 2. *Let Conditions 1-3 of Theorem 3 be fulfilled and let $G = (V, E)$ be the graph computed in Algorithm 1. Let $h', h'' \in A'$ have a conflict with respect to a child h of h' and let $G[h']$ and $G[h'']$ be paths. Then*

1. $\text{side}_g(h'') = \text{side}_h(h'')$ *for all* $g \in A'$ *with* $h' \cap h'' \subseteq g \subseteq h$,
2. $\text{side}_h(h'') \in h''$,
3. $\text{side}_h(h'')$ *is an end vertex of* $G[h']$,
4. $G[h' \setminus h'']$ *is a path, and*
5. $\text{side}_h(h'')$ *is adjacent in* G *to a vertex of* $h'' \setminus h'$.

Proof. We prove the lemma by induction on the sum of the steps in which h' and h'' were considered in Algorithm 1. If h' and h'' had been considered in the first two steps, then at least one of them is a leaf of D' and, hence, h' and h'' have no conflict. So there is nothing to show. Let now h' and h'' be considered in later steps. Let $h'' \in A'$ have a conflict with h' with respect to a child h of h' and let $G[h']$ and $G[h'']$ be paths.

1. + 2. if $h' \cap h'' \in A'$: There is nothing to show if $h = h' \cap h''$. So let h_1 be the child of h with $h_1 \supseteq h' \cap h''$. Then h, h'' have a conflict with respect to h_1. Hence, C_h contains the path h', h'', h_1. By the inductive hypothesis on Property 3, it follows that $\text{side}_{h_1}(h'')$ is an end vertex of $G[h]$, and, especially that h_1 and h share an end vertex. By construction, it follows that $\text{side}_h(h_1)$ is the end vertex of h that is not in h_1. Hence, $\text{side}_h(h'') \in h_1$ and $\text{side}_{h_1}(h'') = \text{side}_h(h'')$. By the inductive hypothesis it follows that $\text{side}_g(h'') = \text{side}_h(h'')$ for $h \cap h'' \subseteq g \subseteq h_1$. Since the labels in $\text{side}_{h' \cap h''}(.)$ are the end vertices of $G[h' \cap h'']$, it follows that $\text{side}_h(h'') \in h' \cap h'' \subset h''$.



(a) augmented Hasse diagram D' (b) tree support G

Fig. 4. Illustration of the proof of Lemma 2.3.

1. + 2. + 5. if $h' \cap h'' \notin A'$: Let $h''_1 \subseteq h''$ be minimal with $h' \cap h'' \subset h''_1$. Since h' and h''_1 overlap there is an edge $\{v, w\} \in E$ such that $v \in h' \cap h''$ and $w \in h''_1 \setminus h'$. We show that $\text{side}_h(h'') = v$.
By Lemma 1 there is a child h_c of h_{vw} that contains $h \cap h_{vw}$. Since $v \in h \cap h_{vw}$ it follows that $w \notin h_c$ and, hence, v is an end vertex of h_c.
Note that by the minimality of h''_1 it follows that $h' \cap h'' \not\subseteq h_{vw}$. Since $G[h''], G[h']$ are paths, it follows that $h_c \subsetneq h$ and, hence, $h_c = h \cap h_{vw}$. Let h_p be minimal with $h_c \subsetneq h_p \subseteq h$. Then h_p, h_{vw} have a conflict with respect to h_c and it follows from the inductive hypothesis on Property 5 that $\text{side}_{h_c}(h_{vw}) = v$. Let h'_c be maximal with $h_c \subseteq h'_c \subsetneq h$. By the inductive hypothesis on Property 1 it follows that $\text{side}_{h'_c}(h_{vw}) = v$. Since h, h_{vw} have a conflict with respect to h'_c it follows by the inductive hypothesis on Property 3 that v is an end vertex of h. In C_h there is the path h'_c, h_{vw}, h', h''. By construction, $\text{side}_h(h'_c)$ is the end vertex of h that is not in h'_c. Hence, $\text{side}_h(h_{vw}) = \text{side}_h(h'') = v$.

3.: Let $v = \text{side}_h(h'')$. By the construction in Algorithm 1, v is an end vertex of $G[h']$ if h' is non-implied. So assume that h' is implied and that v is not an end vertex of $G[h']$. Let $w \in h' \setminus h$ be a neighbor of v in G. By Property 2, it follows that $v \in h''$. Let h_c be the child of h_{vw} that contains $h_{vw} \cap h''$. By the inductive hypothesis on Property 4, it follows that $G[h_{vw} \setminus h'']$ is a path that contains w but not v. Hence, $h_c = h_{vw} \cap h'' = h_{vw} \cap h$.
Let $h'_1, h''_1 \in A'$, respectively, be minimal with $h' \supseteq h'_1 \supsetneq h' \cap h''$ and $h'' \supseteq h''_1 \supsetneq h' \cap h''$. Assume first that $h' \cap h'' \in A'$. Then $C_{h' \cap h''}$ contains the triangle $h_{vw}, h'_1, h''_1, h_{vw}$ and, hence, is not bipartite.
Assume now that $h' \cap h'' \notin A'$. By the already proven part of Property 5 it follows that there is an edge $\{v, x\}$ of G with $x \in h''_1 \setminus h$. We have $h_c = h_{vw} \cap h'' \supseteq h_{vw} \cap h_{vx}$. Further, the child of h_{vx} that contains $h_{vx} \cap h$ equals $h_{vx} \cap h$. Since h''_1 is implied and h_{vx} not, it follows that $h''_1 \neq h_{vx}$ and, hence,

$h_{vx} \not\supseteq h' \cap h''$. Hence, either $h_{vx} \cap h \subseteq h_{vw} \cap h$ or $h_{vw} \cap h \subsetneq h_{vx} \cap h \subsetneq h' \cap h''$. In the first case let $h_1 \in A'$ be minimal with $h_{vw} \cap h \subsetneq h_1 \subseteq h$. Then there is the triangle $h_{vw}, h_{vx}, h_1, h_{vw}$ in $C_{h \cap h_{vw}}$. In the latter case let $h_1 \in A'$ be minimal with $h_{vx} \cap h \subsetneq h_1 \subseteq h$. Then there is the triangle $h_{vw}, h_{vx}, h_1, h_{vw}$ in $C_{h \cap h_{vx}}$.

4.: By the inductive hypothesis $G[h \setminus h'']$ is a path. Further, h and h' share $\text{side}_h(h'') \in h''$ as a common end vertex. By the precondition of the lemma, $G[h']$ is a path. Hence, $G[h' \setminus h'']$ is a path.

5. if $h' \cap h'' \in A'$: If $h \neq h' \cap h''$ let h_1 be the child of h with $h' \cap h'' \subseteq h_1$. By the inductive hypothesis $\text{side}_{h_1}(h'')$ is adjacent in G to a vertex of $h'' \setminus h = h'' \setminus h'$ and by Property 1 $\text{side}_{h_1}(h'') = \text{side}_h(h'')$.
If $h = h' \cap h''$, let $h_1'' \in A'$ be minimal with $h \subsetneq h_1'' \subseteq h''$. Applying Property 3 with h_1'' as "h'" and h' as "h''" reveals that $\text{side}_h(h')$ is an end vertex of $G[h_1'']$. Since $G[h_1'']$ is a path it follows that some vertex of $h_1'' \setminus h$ is adjacent to $\text{side}_h(h'')$. □

Lemma 3. *If Conditions 1-3 of Theorem 3 are fulfilled then all hyperedges in A' induce a path in the graph G constructed in Algorithm 1.*

Proof. Again, we prove the lemma by induction on the step in which h was considered in Algorithm 1. There is nothing to show if h had been considered in the first step. So assume that $h \in A'$ and that $G[h]$ contains a vertex v of degree greater than two.

Let u_1, u_2, u_3 be the first three vertices connected to v in G. Let $h_i = h_{vu_i}, i = 1, 2, 3$. Then h_1, h_2, h_3 are all three contained in h and its intersection contains v. Hence, any two of them have a conflict if and only if one of them is not contained in the other. A case distinction reveals that we wouldn't have appended all three, u_1, u_2 and u_3, to v.

$h_2 = h_3$: Since h_3 contains no vertex of degree higher than two, it follows that $u_1 \notin h_3$, $h_3 \cap h_1 = \{v\}$. Hence, h_1 and h_3 have a conflict with respect to the common child $\{v\}$, contradicting that v is added in the middle of h_3.

$h_1 = h_2$ or $h_1 = h_3$: These cases are analogous to the first case.

$h_1 \subsetneq h_3$: Like in the first case it follows that $u_2 \notin h_3$. Let $h_i', i = 2, 3$ be the child of h_i that contains v. Then h_2 and h_3 have a conflict with respect to $h_i', i = 2, 3$. Since we add the edge $\{v, u_i\}$ to G when we process h_i it follows on one hand that $\text{side}_{h_i'}(h_i) = v$. On the other hand, since h_1 is contained in h_3 and $v \in h_1$ it follows that $h_1 \subseteq h_3'$. Hence, h_3' has more than one vertex. If $h_3' \neq h_3 \cap h_2$ then v is the only end vertex of $G[h_3']$ that is contained in h_2. By Lemma 2 Property 2 it follows that $\text{side}_{h_3'}(h_2) = v$ and hence, $\text{side}_{h_3'}(h_3) \neq v$. If $h_3' = h_3 \cap h_2$ let $v' \neq v$ be the other end vertex of h_2'. Since we know that $\text{side}_{h_2'}(h_2) = v$ it follows that $\text{side}_{h_2'}(h_3) = v'$. Hence, by Lemma 2 Property 1, we can conclude that $\text{side}_{h_3'}(h_3) = v'$. In both cases, we have a contradiction.

$h_1 \subsetneq h_2$ or $h_2 \subsetneq h_3$: These cases are analogous to the third case.

h_1, h_2, h_3 pairwise overlapping: Then $h_1 \cap h_2 = h_2 \cap h_3 = h_1 \cap h_3 = \{v\}$. Hence, Condition 2 of Theorem 3 is not fulfilled. □

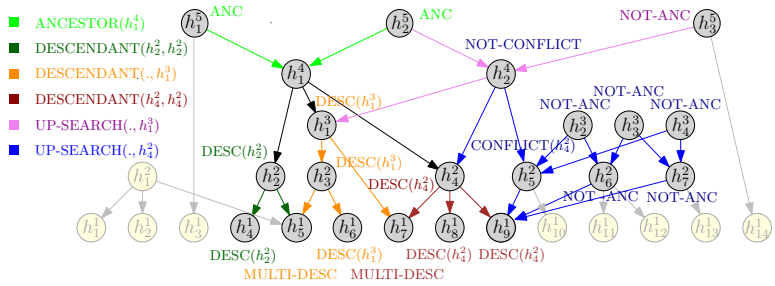


Fig. 5. Computation of the potential conflicts for h_1^4

This completes the proof of Theorem 3. We conclude this section with the following corollary.

Corollary 1. *Algorithm 1 computes a path-based tree support of a hypergraph H if H has a path-based tree support, i.e., if and only if the conditions of Theorem 3 are fulfilled.*

4.3 Conflict Computation and Run Time

In this section we show how to efficiently compute the conflicts and give an upper bound for the run time of testing whether a hypergraph has a path-based tree support and of constructing one, if it exists.

Representing the hyperedges as sorted lists of their elements, all conflicts can be determined straight-forwardly in $\mathcal{O}(n^3(n+m))$ time. In the following, we show how this time bound can be improved.

We first compute candidates for conflicting pairs of hyperedges, which in the case of hypergraphs having a path-based tree support turn out to be a superset of the set of all conflicts. The idea is, that all potential conflicts lie on a path from an ancestor of h to one of h's descendants. The method can be found as pseudocode in Algorithm 2.

We illustrate Algorithm 2 with an example. Figure 5 shows the computation of potential conflicts for the hyperedge h_1^4 of the hypergraph H from Figure 3(a). The different methods are colored. h_5^2 is the only hyperedge that can be in conflict with h_1^4 with respect to a child of h_1^4 and if so, with respect to h_4^2.

Lemma 4. *Let D' be the augmented Hasse diagram of a hypergraph that has a path-based tree support and let h' and h have a conflict with respect to a child h_c of h. Then Algorithm 2 applied to D' and h labels h' with* CONFLICT(h_c).

Proof. Let G be a path-based tree support of a hypergraph and let h' and h have a conflict with respect to a child h_c of h.

1. Let v be the end vertex of $G[h]$ that is contained in h'. Then v and all its ancestors on the path from v to h_c are labeled DESC(h_c) (and not MULTI-DESC).

Algorithm 2: Conflict Computation.

Input : augmented Hasse diagram D' of a hypergraph, vertex h
Output : vertices h' with LABEL(h') = CONFLICT(h_c) for all children h_c of h
Data : there are the following vertex labels
LABEL(h') = ANC iff $h \subsetneq h'$
LABEL(h') = NOT-ANC only if $h \cup h'$ not contained in any source of D'
LABEL(h') = DESC(h_c) iff $h' \subseteq h_c$ for exactly one child h_c of h
LABEL(h') = MULTI-DESC iff h' is contained in more than one child of h
LABEL(h') = NOT-CONFLICT only if $h \cap h'$ not contained in any child of h and $h \cup h'$ contained in some source of D'
LABEL(h') = CONFLICT(h_c) only if $h_c \cap h' \neq \emptyset$ for a child h_c of h and $h \cup h'$ contained in some source of D'

```
ANCESTOR(vertex h') begin
    foreach parent h'' of h' do
        LABEL(h'') ← ANC;
        ANCESTOR(h'');

DESCENDANT(vertex h', vertex h_c) begin
    if LABEL(h') = DESC(h'_c), h_c ≠ h'_c then
        LABEL(h') ← MULTI-DESC;
    else
        LABEL(h') ← DESC(h_c);
    foreach child h'' of h' do
        if LABEL(h'') ≠ MULTI-DESC then
            DESCENDANT(h'', h_c);

UP-SEARCH(vertex h', vertex h_c) begin
    foreach parent h'' of h' do
        if LABEL(h'') ∈ {∅, CONFLICT(h'_c), h'_c ≠ h_c} then
            UP-SEARCH(h'', h_c);
        if LABEL(h') = CONFLICT(h'_c), h_c ≠ h'_c then
            LABEL(h') ← NOT-CONFLICT;
        else if LABEL(h') ≠ DESC(h_c) then
            if LABEL(h'') ∈ {CONFLICT(h_c), ANC, NOT-CONFLICT} then
                LABEL(h') ← CONFLICT(h_c);
            if LABEL(h') ≠ CONFLICT(h_c) then
                LABEL(h') ← NOT-ANC;

begin
    Clear all labels;
    LABEL(h) ← NOT-CONFLICT;
    ANCESTOR(h);
    foreach child h_c of h do
        DESCENDANT(h_c, h_c);
    foreach vertex h' of D' with LABEL(h') ∈ {DESC(h_c); h_c child of h} do
        UP-SEARCH(h', h_c);
```

2. If there was a descendant of h' labeled $\text{DESC}(h'_c)$ for a child $h'_c \neq h_c$ of h, then h_c does not contain $h \cap h'$ contradicting that h and h' have a conflict with respect to h_c.

Hence, Algorithm 2 labels h' with $\text{CONFLICT}(h_c)$. □

Theorem 4. *It can be tested in $\mathcal{O}(n^3m)$ time whether a hypergraph has a path-based tree support and if so such a support can be constructed within the same time bounds.*

Proof. Let H be a hypergraph. First test in linear time whether there is a tree support for H [13]. Let D' be the augmented Hasse diagram of H. The method works in four steps.

1. Start with an empty array conflict indexed with pairs of inner vertices of D'. Set $\text{conflict}_{h,h'} \leftarrow h_c$ if and only if h' is labeled $\text{CONFLICT}(h_c)$ in Algorithm 2 applied to D' and h.
2. For each pair h, h' of inner vertices of D' test whether $\text{conflict}_{h,h'}$ contains $h \cap h'$. Otherwise set $\text{conflict}_{h,h'} \leftarrow \emptyset$. Now, if H has a path-based tree support then h, h' has a conflict with respect to the child h_c of h if and only if $h_c = \text{conflict}_{h,h'}$.
3. Apply Algorithm 1 to compute a support G. If the algorithm stops without computing a support then H does not have a path-based tree support.
4. Test whether every hyperedge induces a path in G. If not, H does not have a path-based tree support.

D' has $\mathcal{O}(n+m)$ vertices, $\mathcal{O}(n^2+nm)$ edges and can be computed in $\mathcal{O}(n^3m)$ time if H has a tree support [3]. Algorithm 2 visits every edge of D' at most twice and, hence, runs in $\mathcal{O}(n^2+nm)$ time for each of the $\mathcal{O}(n)$ inner vertices of D'.

We may assume that the hyperedges are given as sorted lists of their elements. If not given in advance, these lists could straight forwardly be computed from D' in $\mathcal{O}(n^3+mn^2)$ time by doing a graph search from each leaf. Now, for each of the $\mathcal{O}(n^2)$ pairs h, h' of inner vertices it can be tested in $\mathcal{O}(n)$ time whether $\text{conflict}_{h,h'}$ contains $h \cap h'$.

The sum of the sizes of all conflict graphs is in $\mathcal{O}(n^2)$. Hence, Algorithm 1 runs in $\mathcal{O}(n^2+mn)$ time. For each of the $\mathcal{O}(m)$ hyperedges h it can be tested in $\mathcal{O}(n)$ time, whether $G[h]$ is a path. Hence, the overall run time is dominated by the computation of the augmented Hasse diagram and is in $\mathcal{O}(n^3m)$. □

5 Conclusion

We have introduced path-based supports for hypergraphs. Hence, as a new model, we considered a restriction on the appearance of those subgraphs of a support that are induced by the hyperedges. We have shown that it is $\mathcal{NP}$-complete to find the minimum number of edges of a path-based support or to decide whether there is a planar path-based support. Further, we characterized

those hypergraphs that have a path-based tree support and we gave an algorithm that computes a path-based tree support in $\mathcal{O}(n^3m)$ run time if it exists. Our algorithm completed the paths for the hyperedges in the order in which they appeared in a reversed topological ordering of the augmented Hasse diagram. To connect the subpaths in the right order, we introduced a conflict graph for each hyperedge h and colored the vertices of this conflict graph with the end vertices of the path induced by h.

References

1. C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of the Association for Computing Mashinery*, 30(4):479–513, 1983.
2. U. Brandes, S. Cornelsen, B. Pampel, and A. Sallaberry. Hypergraphs and outerplanarity. In this volume.
3. K. Buchin, M. van Kreveld, H. Meijer, B. Speckmann, and K. Verbeek. On planar supports for hypergraphs. In D. Eppstein and E. R. Gansner, editors, *Proceedings of the 17th International Symposium on Graph Drawing (GD 2009)*, Lecture Notes in Computer Science. Springer, 2010.
4. C. Bujtás and Z. Tuza. Color-bounded hypergraphs, II: Interval hypergraphs and hypertrees. *Discrete Mathematics*, 309:6391–6401, 2009.
5. J. Flower, A. Fish, and J. Howse. Euler diagram generation. *Journal on Visual Languages and Computing*, 19(6):675–694, 2008.
6. D. S. Johnson, S. Krishnan, J. Chhugani, S. Kumar, and S. Venkatasubramanian. Compressing large boolean matrices using reordering techniques. In M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer, editors, *Proceedings of the 13th International Conference on Very Large Data Bases (VLDB '04)*, pages 13–23. Morgan Kaufmann, 2004.
7. D. S. Johnson and H. O. Pollak. Hypergraph planarity and the complexity of drawing Venn diagrams. *Journal of Graph Theory*, 11(3):309–325, 1987.
8. M. Kaufmann, M. van Kreveld, and B. Speckmann. Subdivision drawings of hypergraphs. In I. G. Tollis and M. Patrignani, editors, *Proceedings of the 16th International Symposium on Graph Drawing (GD 2008)*, volume 5417 of *Lecture Notes in Computer Science*, pages 396–407. Springer, 2009.
9. E. Korach and M. Stern. The clustering matroid and the optimal clustering tree. *Mathematical Programming, Series B*, 98:385 – 414, 2003.
10. D. Král', J. Kratochvíl, and H.-J. Voss. Mixed hypercacti. *Discrete Mathematics*, 286:99–113, 2004.
11. M. Nöllenburg. An improved algorithm for the metro-line crossing minimization problem. In D. Eppstein and E. R. Gansner, editors, *Proceedings of the 17th International Symposium on Graph Drawing (GD 2009)*, Lecture Notes in Computer Science, pages 381–392. Springer, 2010.
12. P. Simonetto, D. Auber, and D. Archambault. Fully automatic visualisation of overlapping sets. *Computer Graphics Forum*, 28(3):967–974, 2009.
13. R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, 1984.
14. A. Wolff. Drawing subway maps: A survey. *Informatik-Forschung und Entwicklung*, 22:23–44, 1970.