Path-Based Supports for Hypergraphs

Ulrik Brandes, Sabine Cornelsen, Barbara Pampel, Arnaud Sallaberry

Path-Based Supports for Hypergraphs

Ulrik Brandes[a], Sabine Cornelsen[a], Barbara Pampel[a], Arnaud Sallaberry[b]

[a] *Department of Computer and Information Science, University of Konstanz, Box 67, 78457 Konstanz, Germany*

[b] *CNRS UMR 5800 LaBRI, INRIA Bordeaux - Sud Ouest, Pikko, 351, cours de la Libration, 33405 Talence Cedex, France*

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 $\mathcal{NP}$-hard to decide whether a path-based support has a monotone drawing, to determine 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.

Keywords: graph algorithm, graph drawing, hypergraph, metro map layout

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.

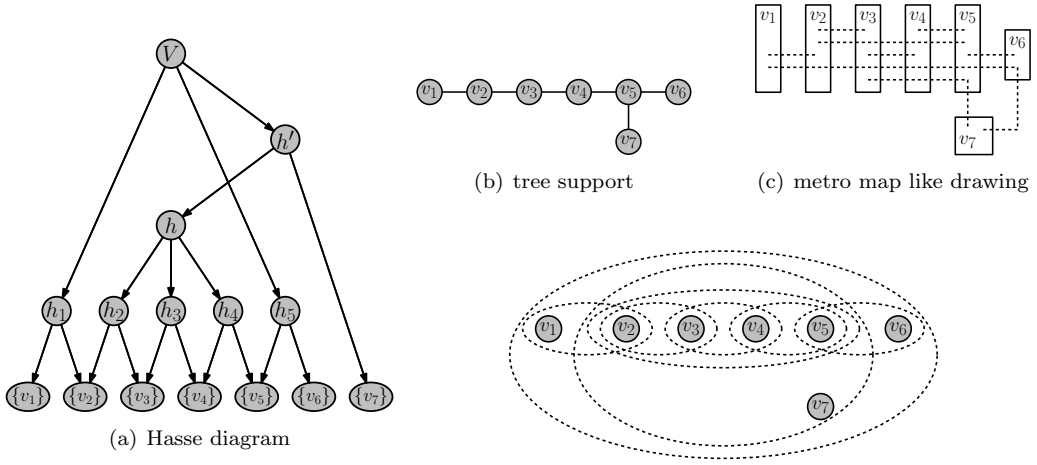
Applications for supports of hypergraphs are, e.g., in hypergraph coloring [2, 3], databases [4], or hypergraph drawing [5, 6, 7, 8]. E.g., see Fig. 1 for an application of a support for designing Euler diagrams. An *Euler diagram* of a hypergraph $H = (V, A)$ is a drawing of H in the plane in which 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. [9, 8].

Recently, many papers have been devoted to the problem of deciding which classes of hypergraphs admit what kind of supports. It can be tested in linear time whether a hypergraph has a support that is a tree [10], a path or a cycle [7]. It can be decided in polynomial time whether a hypergraph has a tree support with bounded degrees [7] or a cactus support [11]. A minimum weighted tree support can be computed in polynomial time [12]. It is $\mathcal{NP}$-complete to decide whether a hypergraph has a planar support [5], a compact support [5, 6] or a 2-outerplanar support [7]. A support with the minimum

Email addresses: Ulrik.Brandes@uni-konstanz.de (Ulrik Brandes), Sabine.Cornelsen@uni-konstanz.de (Sabine Cornelsen), Barbara.Pampel@uni-konstanz.de (Barbara Pampel), arnaud.sallaberry@labri.fr (Arnaud Sallaberry)

A preliminary version of this paper was presented at the 21st International Workshop on Combinatorial Algorithms (IWOCA 2010) and appears in the corresponding proceedings [1].

Corresponding author phone: +49 7531 88 4375, fax: +49 7531 88 3577.



(a) Hasse diagram

(b) tree support

(c) metro map like drawing

(d) Euler diagram

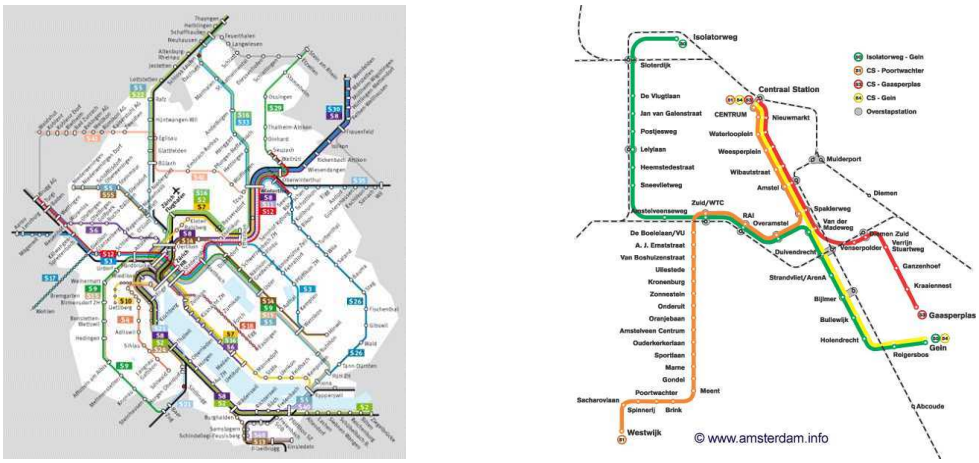
Figure 1: Three representations of the hypergraph $H = (V, A)$ with hyperedges $h_1 = \{v_1, v_2\}$, $h_2 = \{v_2, v_3\}$, $h_3 = \{v_3, v_4\}$, $h_4 = \{v_4, v_5\}$, $h_5 = \{v_5, v_6\}$, $h = \{v_2, v_3, v_4, v_5\}$, $h' = \{v_2, v_3, v_4, v_5, v_7\}$, and $V = \{v_1, \ldots, v_7\}$.

number of edges can be computed in polynomial time if the hypergraph is closed under intersections [7]. If the set of hyperedges is closed under intersections and differences, it can be decided in polynomial time whether the hypergraph has a planar or outerplanar support [11].

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 definition was motivated by 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, Fig. 3 for an example of natural sciences drawn in the metro map anthology, and Fig. 1(c) and 6(f) for a representation of some hyperedges in such a metro map like drawing. For metro map layout algorithms see, e.g., [13, 14].

We briefly consider monotone, planar, 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 discuss monotone path-based supports in Section 3 and mention that finding a minimum path-based support or deciding whether there is a planar path-based support, respectively, is $\mathcal{NP}$-hard. 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.



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

Figure 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.

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 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 $h_1, h_2 \in A$. We say that two hyperedges h_1, h_2 *overlap* if $h_1 \cap h_2 \neq \emptyset$, $h_1 \not\subseteq h_2$, and $h_2 \not\subseteq h_1$. A hypergraph $H = (V, A)$ is *connected* if for any pair of vertices $v, w \in V$ there is a sequence of hyperedges $h_1, \ldots, h_\ell \in A$ such that $v \in h_1, w \in h_\ell$, and $h_i \cap h_{i+1} \neq \emptyset, i = 1, \ldots \ell - 1$.

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. Path-Based Supports

In a *metro map like drawing* of a hypergraph vertices are drawn as disjoint simple closed regions in the plane and each hyperedge h is drawn as a curve $\mathcal{C}_h$ with the end points within the regions of different vertices of h, visiting the region of every vertex of h exactly once, not visiting the vertices not in h, and such that the pieces of $\mathcal{C}_h$ within the region of a vertex or between two such regions are simple. A *path-based support* of a hypergraph $H = (V, A)$ is a graph G such that $G[h]$ contains a spanning path for every hyperedge $h \in A$.

On one hand, a metro map like drawing of a hypergraph $H = (V, A)$ induces a path-based support $G = (V, E)$ of H: For a hyperedge $h \in A$ let $p_h : v_1, \ldots, v_{|h|}$ be the

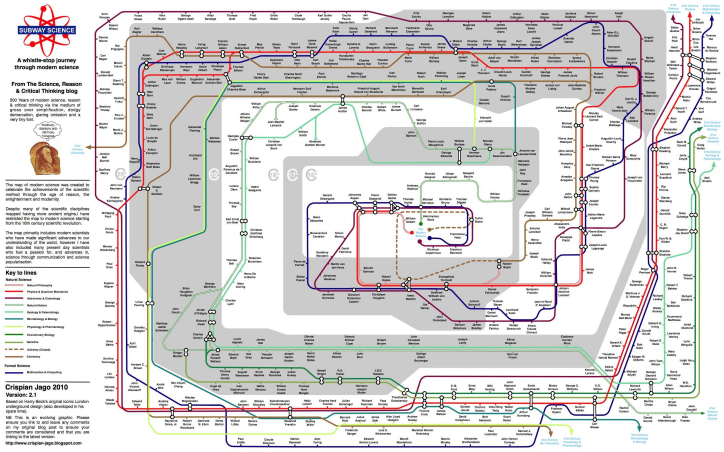


Figure 3: A map of modern science (`www.crispian.net`).

sequence of vertices of h in the order in which they are visited by the curve representing h. Starting with an empty set E add for every hyperedge $h \in A$ with $p_h : v_1, \ldots, v_{|h|}$ the edges $\{v_{i-1}, v_i\}, i = 2, \ldots, |h|$ to E. On the other hand, if we have a path-based support G of H and we fix for every hyperedge $h \in A$ a spanning path p_h of $G[h]$ then this induces a metro map like drawing of H.

In order to have a readable metro map like drawing of a hypergraph it is typically desirable to draw any curve representing a hyperedge without self intersection or even monotone.

3.1. Monotone Path-Based Supports

A *drawing* of a graph is a mapping of each vertex to a distinct point in the plane and of each edge to a simple curve between the image of its adjacent vertices not containing the image of any other vertex. In a straight-line drawing of a graph each edge is drawn as a line segment. Given a drawing of G, a path p of G is *monotone* with respect to a straight line ℓ – called the *axis of monotonicity* – if every line perpendicular to ℓ intersects the drawing of p in at most one point. Note that a path p in a straight-line drawing is monotone with respect to the axis ℓ if and only if the orthogonal projections of the vertices of p on ℓ appear along ℓ in the order induced by p.

Let $G = (V, E)$ be a path-based support of a hypergraph $H = (V, A)$. A drawing of G is *monotone* with respect to H if for each hyperedge $h \in A$ there is a spanning path p_h of $G[h]$ and a straight line ℓ_h such that p_h is monotone with respect to the axis ℓ_h. G is a *monotone* path-based support of H if G has a monotone drawing with respect to H.

Remark 1. *If G has a monotone drawing with respect to a hypergraph H then G has a straight-line drawing that is monotone with respect to H with the same axes of monotonicity.*

Proof. Let a drawing $\mathcal{D}$ of G that is monotone with respect to $H = (V, A)$ be given and let $p_h, h \in A$ be a spanning path of $G[h]$ that is monotone with respect to the axis ℓ_h. If for each edge $\{v, w\}$ of G the line segment between v and w does not contain any vertex of G other than v or w then the straight-line drawing of G in which the vertices are mapped to the same points as in $\mathcal{D}$ is monotone with respect to H.

Consider now for two vertices v, w in a hyperedge h the distances $\text{dist}_h(v, w)$ between the orthogonal projections of v and w to ℓ_h. Let Δ be the minimum of all distances $\text{dist}_h(v, w)$ over all $h \in A$ and $v, w \in h$ with $v \neq w$. Let $0 < \varepsilon \leq \Delta/3$.

Consider now the vertices of V in an arbitrary order $v_1, \ldots, v_n, n = |V|$. For $k = 1, \ldots, n$, we can now place v_k on the circle with radius ε around the position of v_k in $\mathcal{D}$ but not on the intersection with the line through the already fixed drawings of v_i and v_j, $1 \leq i < j < k$. The corresponding straight-line drawing is monotone with respect to H with the axes of monotonicity $\ell_h, h \in A$. □

Remark 2. *Not every path based support of a hypergraph is monotone.*

PROOF. Consider the following hypergraph. Let $I = \{(i, j, k, \ell);\ 1 \leq i < j \leq 5, 1 \leq k < \ell \leq 5, i < k, \{i, j\} \cap \{k, \ell\} = \emptyset\}$ be an index set representing unordered pairs of disjoint edges of the complete graph K_5. Let $V_I = \{v_i;\ i = 1, \ldots, 5\} \cup \{v_{i,j,k,\ell,x};\ (i, j, k, \ell) \in I, x = 1, \ldots, 3\}$, let $h_{ijk\ell} = \{v_i, v_{i,j,k,\ell,1}, v_j, v_{i,j,k,\ell,2}, v_k, v_{i,j,k,\ell,3}, v_\ell\}$, $(i, j, k, \ell) \in I$, let $A_I = \{h_{ijk\ell};\ (i, j, k, \ell) \in I\}$, and let $H_I = (V_I, A_I)$. Let E contain the edges $\{v_i, v_{i,j,k,\ell,1}\}$, $\{v_{i,j,k,\ell,1}, v_j\}$, $\{v_j, v_{i,j,k,\ell,2}\}$, $\{v_{i,j,k,\ell,2}, v_k\}$, $\{v_k, v_{i,j,k,\ell,3}\}$, $\{v_{i,j,k,\ell,3}, v_\ell\}$ for $(i, j, k, \ell) \in I$. The resulting path-based support $G = (V, E)$ of H_I is shown in Fig. 4(a). Note that $G[h_{ijk\ell}]$ is a path for any hyperedge $h_{ijk\ell} \in A$ visiting the vertices v_i, v_j, v_k, v_ℓ in this order. Consider now any drawing of G. Since a K_5 is not planar, there are two straight line segments $\overline{v_i v_j}, \overline{v_k, v_\ell}, (i, j, k.\ell) \in I$ that intersect. Hence, the path $G[h_{ijk\ell}]$ cannot be drawn monotonously. □

Remark 3. *Every hypergraph has a monotone path-based support.*

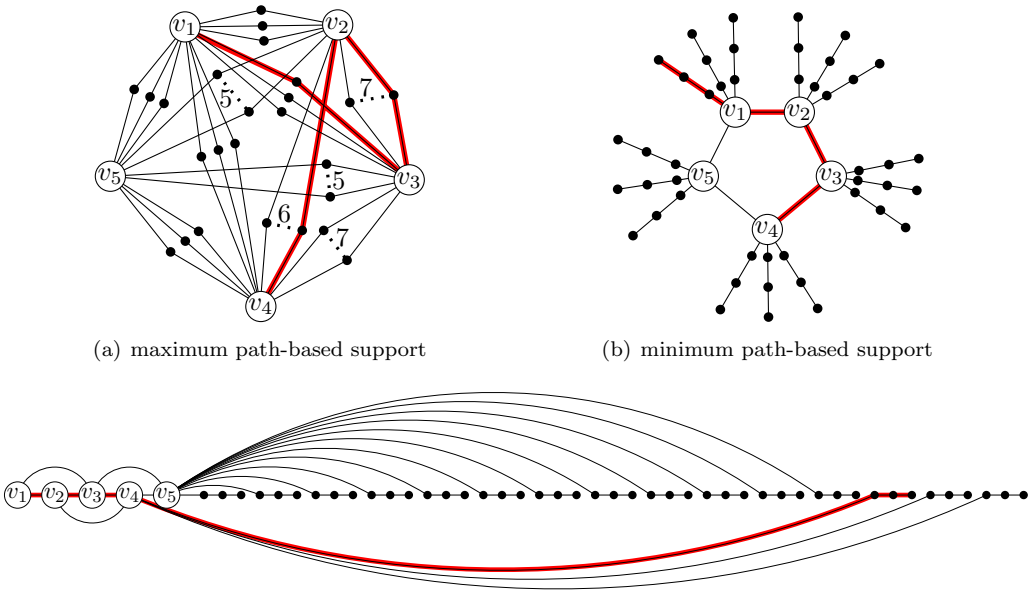
PROOF. Order the vertices of $H = (V, A)$ with respect to an arbitrary ordering $<$. The *support $G_< = (V, E_<)$ of H with respect to the ordering $<$* is constructed as follows. For each hyperedge $\{v_1, \ldots, v_k\} \in A$ with $v_1 < \cdots < v_k$ the edge set $E_<$ contains the edges $\{v_{i-1}, v_i\}, i = 1, \ldots, k$. Assume now that in a drawing of $G_<$ the x-value of a vertex v is smaller than the x-value of the vertex w if $v < w$ and that the edges are drawn monotonously in x-direction. Then for each hyperedge $h = \{v_1, \ldots, v_k\} \in A$ with $v_1 < \cdots < v_k$ the path $p_h : v_1, \ldots, v_k$ is drawn monotonously with respect to the x-axis. See Fig 4(c) for an example. □

Note that the problem of deciding whether a given support is a support with respect to an ordering and if so, finding such an ordering, is closely related to the betweenness problem [15].

Theorem 1. *Given a support G of a hypergraph H*

1. *it is $\mathcal{NP}$-hard to decide whether G is a monotone path-based support of H, and*
2. *it is $\mathcal{NP}$-complete to decide whether there exists an ordering $<$ of the vertex set such that G is the support of H with respect to $<$,*

even if G has the minimum number of edges among all supports of H.



(a) maximum path-based support

(b) minimum path-based support

(c) minimum path-based support w.r.t. an ordering

Figure 4: Three different supports for the hypergraph H_I introduced in Section 3.1. The small black vertices are the vertices $v_{\sigma,x}, \sigma \in I, x = 1, 2, 3$. The thick red path indicates the hyperedge h_{1324}.

PROOF.

1. Consider an instance of the *strictly monotone trajectory drawing problem* consisting of a set of paths P on a set of vertices V_t. It is $\mathcal{NP}$-hard to decide whether the vertices can be mapped to points in the plane such that each path is monotone with respect to some axis (one for each path) [16].
   Consider the hypergraph $H = (V, A)$ with V containing V_t and for each path $p \in P$ and each edge $e \in p$ a vertex v_{ep}. The set A contains for each path $p \in P$ a hyperedge $h_p = \bigcup_{\{v,w\} \in p} \{v, v_{\{v,w\}p}, w\}$ as well as the hyperedges $\{v, v_{\{v,w\}p}\}$ and $\{v_{\{v,w\}p}, w\}$ for each edge $\{v, w\} \in p$. The graph $G = (V, E)$ with $E = \bigcup_{p \in P} \bigcup_{e \in p} \{\{v, v_{ep}\};\ v \in e\}$ is a path-based support of H and has the minimum number of edges among all supports of H. G is monotone if and only if P is drawable with each path monotone with respect to some axis.
2. Consider an instance of the *betweenness problem* consisting of a set of vertices V_b and a set of constraints C. Each constraint $c \in C$ consists of a sequence of three vertices. It is $\mathcal{NP}$-complete to decide whether the vertices can be totally ordered such that for each constraint $c = (u, v, w)$ the vertex v is between the vertices u and w [15].
   Consider the hypergraph $H = (V, A)$ with V containing V_b and for each constraint $c \in C$ vertices v_{c2} and v_{c4}. The set A contains for each $c = (v_{c1}, v_{c3}, v_{c5}) \in C$ a hyperedge $h_c = \{v_{c1}, \ldots, v_{c5}\}$ and hyperedges $h_{ci} = \{v_{ci}, v_{c(i+1)}\}$ for $1 \leq i \leq 4$. The graph $G = (V, E)$ with $E = \bigcup_{c \in C} \{h_{ci};\ 1 \leq i \leq 4\}$ is a path-based support of H and has the minimum number of edges among all supports of H.
   There is an ordering $<$ of V such that G is the support of H with respect to $<$ if and only if for each constraint $c = (v_{c1}, v_{c3}, v_{c5}) \in C$ the five vertices in h_v are either ordered $v_{c1} < v_{c2} < v_{c3} < v_{c4} < v_{c5}$ or $v_{c5} < v_{c4} < v_{c3} < v_{c2} < v_{c1}$. Since

the vertices v_{c2} and v_{c4} do not appear in a hyperedge $h_{c'}$ for any constraint $c \neq c'$ it follows that there is an ordering $<$ of V such that G is the support of H with respect to $<$ if and only if V_b can be totally ordered while satisfying all betweenness constraints in C. □

3.2. Minimum Path-Based Supports

Assuming that each hyperedge contains at least one vertex, each hypergraph $H = (V, A)$ has a monotone path-based support $G = (V, E)$ with at most $N - m$ edges. Just take the support $G_<$ with respect to an arbitrary ordering $<$ of the vertex set V. It is, however, $\mathcal{NP}$-hard to find an ordering that minimizes the number of edges among all path-based supports of H with respect to an ordering of the vertex set [17].

Further, note that a path-based support that minimizes the number of edges among all path-based support of a hypergraph H with respect to some ordering of the vertex set might not be a path-based support of H with the minimum number of edges over all path-based supports of H. E.g., consider the hypergraph H_I from the previous section (Fig. 4) or the hypergraph H with hyperedges $\{1, 2, 4\}$, $\{1, 3, 4\}$, and $\{2, 3, 4\}$ for an easier example: the unique minimum path-based support of H is a star centered at 4 which cannot be created from any ordering of the vertex set. The problem of finding a minimum path-based support remains, however, $\mathcal{NP}$-hard.

Theorem 2. *It is $\mathcal{NP}$-hard to minimize the number of edges among all path-based supports (or among all monotone path-based supports) of a hypergraph – even if the hypergraph 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 $K = |E|$. Note that any support of H contains G as a subgraph. Hence, H has a path-based support with at most K edges if and only if G is a path-based support of H which is true if and only if G contains a Hamiltonian path. □

3.3. Planar Path-Based Supports

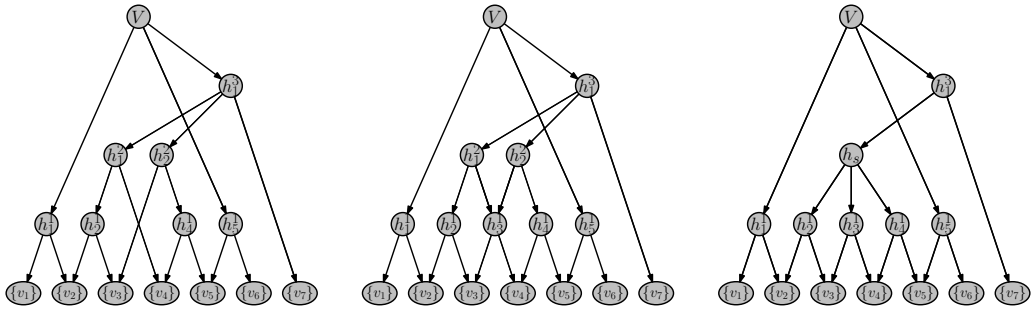
A graph is *planar* if it has a drawing in which no pair of edges intersect but in common end points. For the application of Euler diagram like drawings, planar supports are of special interest. However, like for general planar supports, the problem of testing whether there is a path-based planar support is hard.

Theorem 3. *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 [5] 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, a monotone path-based support [18], and a planar path-based support. Moreover the intersection of any subset of hyperedges induces again a path in a path-based tree support. So far it is known how to decide in linear time whether there is a path-based tree support if $V \in A$ [7].



(a) Hasse diagram D (b) Closure $\overline{D}$ (c) Augmented D'

Figure 5: Illustration of the augmented Hasse Diagram for the hypergraph $H = (V, A)$ indicated in 5(a). $\overline{A} = A \cup \{h_3^1, \{v_2\}, \ldots, \{v_5\}\}$. The two hyperedges h_1^2 and h_2^2 are both implied but no summery edges. They are not present in the augmented Hasse diagram. The summary hyperedge h_s is added to A'.

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 [7]. Note, however, that the number of intersections of any subset of hyperedges could be exponential in the size of the hypergraph.

To construct a tree support of an arbitrary hypergraph $H = (V, A)$, 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 $\overline{H}$: The intersection of two subtrees is again a subtree.

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 might not 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 $\overline{D}$. For an example consider Fig. 5(c). In this example, the hyperedge h_s is a summary hyperedge, h_1^3 and $h_1^1, \ldots, h_5^1$ are non-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 [7] (without explicitly constructing the closure under intersection $\overline{A}$). 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: Otherwise there would be a summary hyperedge between h and intersecting children.

If a tree support $G = (V, E)$ of H exists, it can be constructed as follows [7]. 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 are 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 need 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.

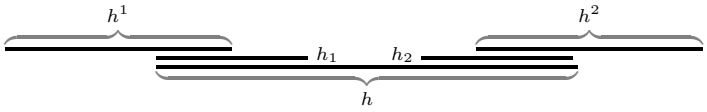
Definiton 1 (Conflicting Hyperedges). *Two overlapping hyperedges $h', h'' \in A'$ have a* conflict *if there is some hyperedge in A' that contains both h' and h''. Two overlapping hyperedges $h', h'' \in A'$ have a* conflict with respect to $h \in A'$ *if h' has a conflict with h'', $h' \cap h'' \subseteq h$ 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 $h \in A'_{h_i}$.*

E.g., consider the hypergraph in Fig. 5(c). Then h_1^3 and h_1^1 are both contained in the hyperedge V and they both contain $\{v_2\}$. Hence, they have a conflict. Further, is the intersection $h_1^3 \cap h_1^1 = \{v_2\}$ contained in the child h_s of h_1^3. Hence, h_1^3 has a conflict with h_1^1 with respect to h_s. Similarly does h_1^3 have a conflict with h_5^1 with respect to h_s and we have on one hand $A'_{h_s} = \{h_1^1, h_2^1, h_1^3\}$. On the other hand does h_s have a conflict with h_1^1 with respect to h_2^1 and with h_5^1 with respect to h_4^1 and we have $A^c_{h_s} = \{h_2^1, h_4^1\}$. Note that there might be hyperedges that have a conflict but not with respect to any of their children. As an example see the hyperedges h_1^4 and h_2^4 in Fig. 6(a). In the lemmas in this section, we will prove that it suffices if the algorithm considers only conflicts with respect to some child.

Assume now that H has a path-based tree support G and let $h', h'' \in A'$ be such that h' and h'' have a conflict with respect to a child h of h''. Since $h' \cup h''$ is contained in a hyperedge it follows that $G[h' \cup h'']$ is the subgraph of a path. Since in addition h' and h'' intersect and $G[h']$ and $G[h'']$ are paths, it follows that $G[h' \cup h'']$ is also a path. Hence, we have the following situation.

Note especially that among the two end vertices of $G[h'']$ exactly one is contained in h' and that this end vertex is also an end vertex of $G[h]$. This yields the following three types of restrictions on the connections of the paths.

1. $G[h' \setminus h]$ and $G[h'' \setminus h]$ must be paths that are attached to different end vertices of $G[h]$.
2. Assume further that h''' does also have a conflict with h'' with respect to h. Then both, $G[h' \setminus h]$ and $G[h''' \setminus h]$, must be appended to the common end vertex of $G[h]$ and $G[h'']$.
3. Assume further that $h_2, h_1 \in A^c_h, h_2 \neq h_1$. 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 common end vertex of $G[h]$ and $G[h_i]$. Hence, $G[h^1 \setminus h]$ and $G[h^2 \setminus h]$ must be appended to different and vertices of $G[h]$.



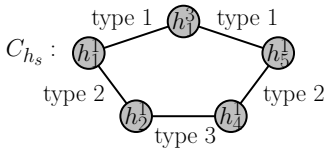
E.g., consider the hypergraph $H = (V, A)$ in Fig. 5(c). Then on one hand, h_1^3 has a conflict with h_1^1 and h_5^1 with respect to h_s. Hence, by the first type of restrictions $G[h_1^1 \setminus h_s]$ and $G[h_5^1 \setminus h_s]$ must be appended to the same end vertex of $G[h_s]$, i.e. the end vertex of $G[h_s]$ to which $G[h_1^3 \setminus h_s]$ is not appended. On the other hand, h_1^1 and h_s have a conflict with respect to h_2^1, while h_5^1 and h_s have a conflict with respect to h_4^1. Hence, by the third type of restrictions it follows that $G[h_1^1 \setminus h_s]$ and $G[h_5^1 \setminus h_s]$ must 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.

Definiton 2 (Conflict Graph). *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. 5(c). Then the conflict graph C_{h_s} contains the edges $\{h_1^3, h_5^1\}$ and $\{h_1^3, h_1^1\}$ of type one, the edges $\{h_2^1, h_1^1\}$ and $\{h_4^1, h_5^1\}$ of type 2 and the edge $\{h_2^1, h_4^1\}$ of type 3. (See the figure on the right.) Hence, C_{h_s} contains a cycle of odd length, reflecting that there is no suitable assignment of the end vertices of $G[h_s]$ to h_1^1, h_5^1 and h_1^3.



Theorem 4. *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 4 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 4 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 the choice of the ordering of the children in Line 8 of Algorithm 1: 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 [7]. Before we show that G is a path-based tree support, we illustrate the algorithm with an example. Consider the hypergraph H in Fig. 6. We show how the algorithm proceeds h_1^5 and all its descendants in D'. For

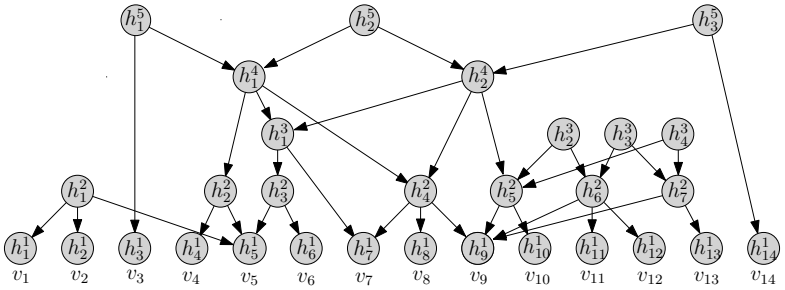
Algorithm 1: Path-based tree support

Input : augmented Hasse diagram D' of a hypergraph $H = (V, A)$
fulfilling the conditions of Theorem 4;
conflict graphs C_h on vertex sets $A'_h \cup A^c_h$, h non-source vertex of D'
Output: path-based tree support $G = (V, E)$ of H
Data : labels $\mathrm{side}_h(h')$
indicating the end vertex of $G[h]$ to which $h' \setminus h$ should be appended

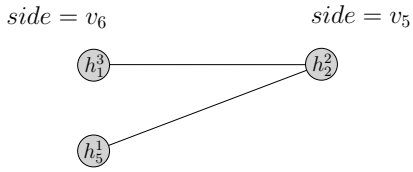
begin
  $E \leftarrow \emptyset$;
  foreach *vertex* h *of* D' *in a reversed topological order of* D' **do**
    if $h = \{v\}$ *for some* $v \in V$ **then**
      foreach *vertex* h' *of* C_h **do**
        $\mathrm{side}_h(h') \leftarrow v$;
    else
      Let $h_1, \ldots, h_k$ be the children of h such that $h_2, \ldots, h_{k-1} \notin A^c_h$;
      if h *is non-implied* **then**
        Let $w_i, v_{i+1}, i = 1, \ldots, k$ be the end vertices of $G[h_i]$ such that
        - $\mathrm{side}_{h_1}(h) = v_2$ if $h \in A'_{h_1}$ and
        - $\mathrm{side}_{h_k}(h) = w_k$ if $h \in A'_{h_k}$;

        Add the edges $\{v_i, w_i\}, i = 2, \ldots, k$ to E;
      else
        Let $w_1 \neq v_{k+1}$ be the end vertices of $G[h]$ such that
        - $v_{k+1} \notin h_1$ and
        - $w_1 \notin h_k$;

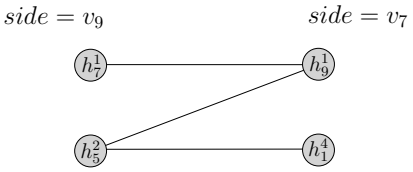
        if $h_1 \in A^c_h$ **then** $\mathrm{side}_h(h_1) \leftarrow v_{k+1}$;
        if $h_k \in A^c_h$ **then** $\mathrm{side}_h(h_k) \leftarrow w_1$;
        Label the remaining vertices of C_h with v_{k+1} or w_1
        such that no two adjacent vertices have the same label;
end



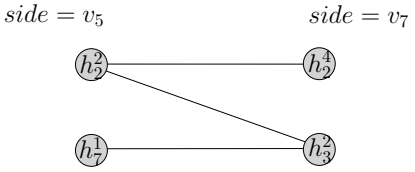
(a) augmented Hasse diagram D' of a hypergraph H



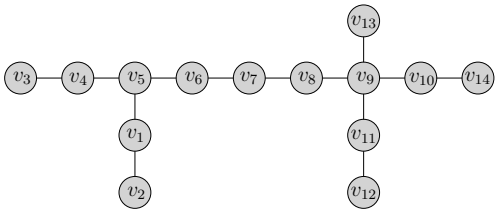
(b) conflict graph of hyperedge h_3^2



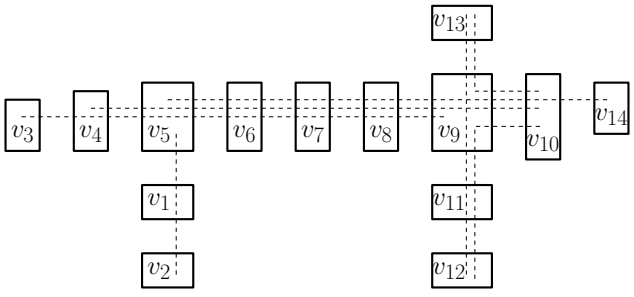
(c) conflict graph of hyperedge h_4^2



(d) conflict graph of hyperedge h_1^3



(e) a path-based tree support of H



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

Figure 6: Illustration of Algorithm 1.

the hyperedges h_3^1, h_4^1, h_6^1, and h_8^1 the conflict graphs are empty. For the other leaves we have

$$\begin{aligned} \text{side}_{h_5^1}(h_2^2) = \text{side}_{h_5^1}(h_3^2) = \text{side}_{h_5^1}(h_1^3) = \text{side}_{h_5^1}(h_2^4) &= v_5, \\ \text{side}_{h_7^1}(h_4^2) = \text{side}_{h_7^1}(h_1^3) &= v_7, \text{ and} \\ \text{side}_{h_9^1}(h_4^2) = \text{side}_{h_9^1}(h_1^4) = \text{side}_{h_9^1}(h_5^2) = \text{side}_{h_9^1}(h_6^2) = \text{side}_{h_9^1}(h_7^2) &= v_9. \end{aligned}$$

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 $\text{side}_{h_2^2}(h_5^1) = v_4$, the assignment of side in $C_{h_3^2}$ is illustrated in Fig. 6(b). h_4^2 has a conflict with respect to the children h_7^1 and h_9^1. Hence, we add edges $\{v_7, v_8\}$ and $\{v_8, v_9\}$ to G. The conflict graph of h_4^2 is shown in Fig. 6(c). When operating h_1^3 we can choose $h_1 = h_3^2$ and $h_2 = h_7^1$, since $\text{side}_{h_3^2}(h_1^3) = v_6$ and $\text{side}_{h_7^1}(h_1^3) = v_7$. We add the edge $\{v_6, v_7\}$ to G. The conflict graph $C_{h_1^3}$ is shown in Fig. 6(d). 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 5. *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 $h \in A'$. □

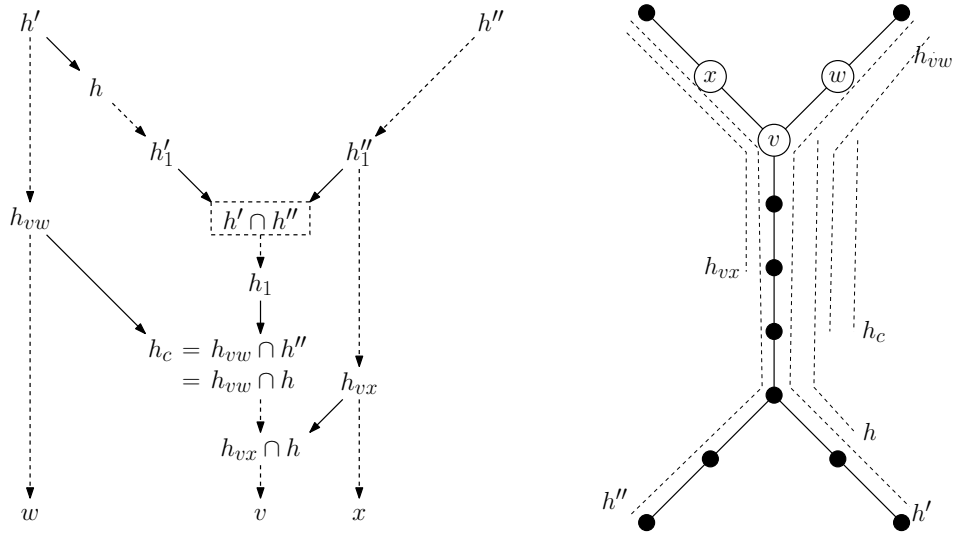
For an edge $\{v, w\}$ of G let h_{vw} be the intersection of all hyperedges of A' that contain v and w. Note that h_{vw} is not implied since v and w are contained in different children of h_{vw} in $\overline{D}$ and $\{v, w\}$ is an edge of the tree support G of $\overline{H}$. Hence, $h_{vw} \in A'$.

Lemma 6. *Let Conditions 1-3 of Theorem 4 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



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

Figure 7: Illustration of the proof of Lemma 6.3.

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 $\text{side}_h(h'') \in h' \cap h'' \subset h''$.

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 5 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 $h'_1, h''_1 \in A'$ be minimal with $h' \supseteq h'_1 \supsetneq h' \cap h''$ and $h'' \supseteq h''_1 \supsetneq h' \cap h''$, respectively. 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 7. *If Conditions 1-3 of Theorem 4 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 6 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 6 Property 1 we can conclude that $\text{side}_{h'_3}(h_3) = v'$. In both cases we have a contradiction.

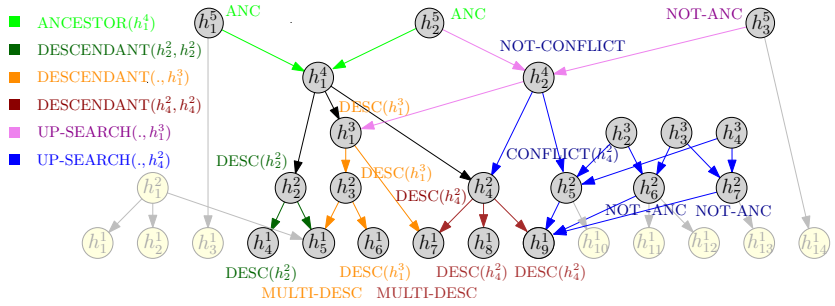


Figure 8: Illustration of Algorithm 2. Computation of the potential conflicts for h_1^4

$h_1 \subsetneq h_2$ **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 4 is not fulfilled. □

This completes the proof of Theorem 4. We conclude this section with the following corollary.

Corollary 8. *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 4 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, if it exists, of constructing one.

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. First, all ancestors of h are marked in Procedure ANCESTOR(h). The procedures DESCENDANT(h_c, h_c), h_c child of h label those descendants of a child h_c of h with DESC(h_c) that are not descendants of any other child of h. Finally, after all calls of UP-SEARCH a hyperedge h' is labeled CONFLICT(h_c) if and only if h' is (1) a descendant of an ancestor of h, but neither a descendant nor an ancestor of h, (2) an ancestor of a descendant of the child h_c of h that is not a descendant of another child of h, and (3) if all descendants h' are descendants of h_c or of at least two children of h. This implies especially that h and h' have a conflict, if h' is labeled CONFLICT(h_c).

Before we show that h' is labeled CONFLICT(h_c) if h and h' have a conflict with respect to h_c, we illustrate Algorithm 2 with an example. Figure 8 shows the computation of potential conflicts for the hyperedge h_1^4 of the hypergraph H from Figure 6(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.

Algorithm 2: Conflict Computation.

Input : augmented Hasse diagram D' of a hypergraph; vertex h
Output: for each child h_c of h the vertices h' with $\textsc{label}(h') = \textsc{conflict}(h_c)$
Data : there are the following vertex labels
$\textsc{label}(h') = \textsc{anc}$ iff $h \subsetneq h'$
$\textsc{label}(h') = \textsc{not-anc}$ only if $h \cup h'$ not contained in any source of D'
$\textsc{label}(h') = \textsc{desc}(h_c)$ iff $h' \subseteq h_c$ for exactly one child h_c of h
$\textsc{label}(h') = \textsc{multi-desc}$ iff h' is contained in more than one child of h
$\textsc{label}(h') = \textsc{not-conflict}$ only if $h \cap h'$ not contained in any child of h and $h \cup h'$ contained in some source of D'
$\textsc{label}(h') = \textsc{conflict}(h_c)$ 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'');
end

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);
end

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;
end

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_d of D' with LABEL(h_d) ∈ {DESC(h_c); h_c child of h} do
        UP-SEARCH(h_d, h_c);
end
```

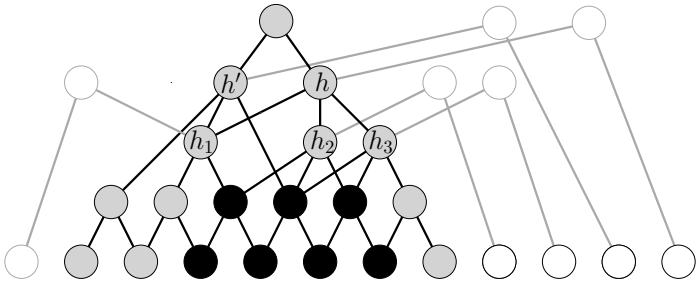


Figure 9: h' and h do not have a conflict with respect to any child of h but the label of h' is CONFLICT(h_1) in the end of Algorithm 2 applied to h. All descendants of h_2 (black vertices) are labeled MULTI-DESC.

Lemma 9. *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 the label of h' in the end of Algorithm 2 applied to D' and h is* 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).
2. If there was a descendant of h' labeled DESC(h'_c) for a child $h'_c \neq h_c$ of h, then h' contains a vertex of h that is not a contained in h_c. Hence, h_c does not contain $h \cap h'$, contradicting that h and h' have a conflict with respect to h_c. Hence, Algorithm 2 does not label h' with NOT-CONFLICT.

For a path P in D' from h' to v let h_P be the first vertex on P that is labeled DESC(h_c). Assume that among all such vertices h_P is the one to which the procedure UP-SEARCH of Algorithm 2 is applied first. Then UP-SEARCH(h_P, h_c) labels h' with CONFLICT(h_c). □

Note, however, that the converse of Lemma 9 is not true. More precisely, if h' is labeled CONFLICT(h_c) in the end of Algorithm 2 applied to h then indeed do h and h' have a conflict, but h_c does not have to contain $h \cap h'$. The reason for this is that all descendants of h' that are no descendants of h_c are descendants of several children of h and, hence, labeled MULTI-DESC. Fig. 9 shows an example.

Theorem 10. *It can be tested in $\mathcal{O}(n^3 m)$ 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 [10]. 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 CONFLICT(h_c) in the end of 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 [7]. 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 discussed that monotone path-based supports are desirable. We have shown that it is $\mathcal{NP}$-hard to decide whether a given path-based support is monotone or to find a path-based support with the minimum number of edges. Further, it is $\mathcal{NP}$-complete to decide whether there is a planar path-based support. As a main result, 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 appear in a reversed topological ordering of the augmented Hasse diagram. To connect these subpaths in the right order, we introduced a conflict graph for each hyperedge h and colored its vertices with the end vertices of the path induced by h.

References

[1] U. Brandes, S. Cornelsen, B. Pampel, A. Sallaberry, Path-based supports for hypergraphs, in: C. Iliopoulos, W. Smyth (Eds.), Proceedings of the 21st International Workshop on Combinatorial Algorithms (IWOCA 2010), Vol. 6460 of Lecture Notes in Computer Science, Springer, 2011, pp. 20–33.

[2] D. Král', J. Kratochvíl, H.-J. Voss, Mixed hypercacti, Discrete Mathematics 286 (2004) 99–113.

[3] C. Bujtás, Z. Tuza, Color-bounded hypergraphs, II: Interval hypergraphs and hypertrees, Discrete Mathematics 309 (2009) 6391–6401.

[4] C. Beeri, R. Fagin, D. Maier, M. Yannakakis, On the desirability of acyclic database schemes, Journal of the Association for Computing Machinery 30 (4) (1983) 479–513.

[5] D. S. Johnson, H. O. Pollak, Hypergraph planarity and the complexity of drawing Venn diagrams, Journal of Graph Theory 11 (3) (1987) 309–325.

[6] M. Kaufmann, M. van Kreveld, B. Speckmann, Subdivision drawings of hypergraphs, in: I. G. Tollis, M. Patrignani (Eds.), Proceedings of the 16th International Symposium on Graph Drawing (GD 2008), Vol. 5417 of Lecture Notes in Computer Science, Springer, 2009, pp. 396–407.

[7] K. Buchin, M. van Kreveld, H. Meijer, B. Speckmann, K. Verbeek, On planar supports for hypergraphs, in: D. Eppstein, E. R. Gansner (Eds.), Proceedings of the 17th International Symposium on Graph Drawing (GD 2009), Vol. 5849 of Lecture Notes in Computer Science, Springer, 2010, pp. 345–356.

[8] P. Simonetto, D. Auber, D. Archambault, Fully automatic visualisation of overlapping sets, Computer Graphics Forum 28 (3) (2009) 967–974.

[9] J. Flower, A. Fish, J. Howse, Euler diagram generation, Journal on Visual Languages and Computing 19 (6) (2008) 675–694.

[10] R. E. Tarjan, 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) (1984) 566–579.

[11] U. Brandes, S. Cornelsen, B. Pampel, A. Sallaberry, Blocks of hypergraphs applied to hypergraphs and outerplanarity, in: C. Iliopoulos, W. Smyth (Eds.), Proceedings of the 21st International Workshop on Combinatorial Algorithms (IWOCA 2010), Vol. 6460 of Lecture Notes in Computer Science, Springer, 2011, pp. 201–211.

[12] E. Korach, M. Stern, The clustering matroid and the optimal clustering tree, Mathematical Programming, Series B 98 (2003) 385 – 414.

[13] M. Nöllenburg, An improved algorithm for the metro-line crossing minimization problem, in: D. Eppstein, E. R. Gansner (Eds.), Proceedings of the 17th International Symposium on Graph Drawing (GD 2009), Vol. 5849 of Lecture Notes in Computer Science, Springer, 2010, pp. 381–392.

[14] A. Wolff, Drawing subway maps: A survey, Informatik-Forschung und Entwicklung 22 (1970) 23–44.

[15] J. Opatrny, Total ordering problem, SIAM Journal on Computing 8 (1) (1979) 111–114.

[16] B. Pampel, Constrained graph drawing, Ph.D. thesis, University of Konstanz, to appear (2011).

[17] D. S. Johnson, S. Krishnan, J. Chhugani, S. Kumar, S. Venkatasubramanian, Compressing large boolean matrices using reordering techniques, in: M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, K. B. Schiefer (Eds.), Proceedings of the 13th International Conference on Very Large Data Bases (VLDB '04), Morgan Kaufmann, 2004, pp. 13–23.

[18] P. Angelini, E. Colasante, G. Di Battista, F. Frati, M. Patrignani, Monotone drawings of graphs, in: U. Brandes, S. Cornelsen (Eds.), Proceedings of the 18th International Symposium on Graph Drawing (GD 2010), Vol. 6502 of Lecture Notes in Computer Science, Springer, 2011, pp. 13–24.