Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 15, no. 4, pp. 533–549 (2011)

On Planar Supports for Hypergraphs

Kevin Buchin[1] *Marc van Kreveld*[2] *Henk Meijer*[3]
Bettina Speckmann[1] *Kevin Verbeek*[1]

[1]Department of Mathematics and Computer Science,
TU Eindhoven, The Netherlands
[2]Department of Information and Computing Sciences,
Utrecht University, The Netherlands
[3]Roosevelt Academy, Middelburg, The Netherlands

Abstract

A graph G is a *support* for a hypergraph $H = (V, \mathcal{S})$ if the vertices of G correspond to the vertices of H such that for each hyperedge $S_i \in \mathcal{S}$ the subgraph of G induced by S_i is connected. G is a *planar support* if it is a support and planar. Johnson and Pollak [11] proved that it is NP-complete to decide if a given hypergraph has a planar support. In contrast, there are lienar time algorithms to test whether a given hypergraph has a planar support that is a path, cycle, or tree. In this paper we present an efficient algorithm which tests in polynomial time if a given hypergraph has a planar support that is a tree where the maximal degree of each vertex is bounded. Our algorithm is constructive and computes a support if it exists. Furthermore, we prove that it is already NP-hard to decide if a hypergraph has a 2-outerplanar support.

Submitted:	Reviewed:	Revised:	Accepted:
January 2010	January 2011	June 2011	August 2011
	Final:	Published:	
	September 2011	September 2011	

Article type:
Regular Paper

Communicated by:
M. Kaufmann

E-mail addresses: k.a.buchin@tue.nl (Kevin Buchin) m.j.vankreveld@uu.nl (Marc van Kreveld) h.meijer@roac.nl (Henk Meijer) speckman@win.tue.nl (Bettina Speckmann) k.a.b.verbeek@tue.nl (Kevin Verbeek)

1 Introduction

A *hypergraph* $H = (V, \mathcal{S})$ is a generalization of a graph, where V is a set of elements or vertices and $\mathcal{S}$ is a set of non-empty subsets of V, called *hyperedges* [3]. The set $\mathcal{S}$ of hyperedges is a subset of the powerset of V. Hypergraphs are not as common as graphs, but there are several application areas where they occur. For example, there is a natural correspondence between hypergraphs and database schemata in relational databases, with vertices corresponding to attributes and hyperedges to relations (e.g., see [2]). Further applications include VLSI design [15], computational biology [14], and social networks [7].

There is no single "standard" method of drawing hypergraphs, comparable to the point-and-arc drawings for graphs. In this paper we focus on a set of decision problems which are motivated by *subdivision drawings* of hypergraphs as proposed by Kaufmann et al. [12]. In a subdivision drawing each vertex corresponds uniquely to a face of a planar subdivision and, for each hyperedge, the union of the faces corresponding to the vertices incident to that hyperedge is connected. For example, vertex-based Venn diagrams [11] and concrete Euler diagrams [9] are both subdivision drawings.

A graph G is a *support* for a hypergraph $H = (V, \mathcal{S})$ if the vertices of G correspond to the vertices of H such that for each hyperedge $S_i \in \mathcal{S}$ the subgraph of G induced by S_i is connected. We say that S_i is connected in G. G is a *planar support* if it is a support and planar. Intuitively, a planar support is a subgraph of the dual graph of a subdivision drawing of H. Subdivisions and their dual graphs have been studied extensively and there are several methods that can turn a planar support into a dual subdivision.

Johnson and Pollak [11] proved that it is NP-complete to decide if a given hypergraph has a planar support. In contrast, there are linear time algorithms that decide whether a given hypergraph has a planar support that is either a path, a cycle, or a tree. We discuss these results in some detail in Section 2. Brandes *et al.* [5] show how to test in polynomial time whether a given hypergraph has a cactus support, that is, a support that is a tree of edges and cycles. The authors also show how to test in polynomial time whether a hypergraph that is closed under intersections and differences has an outerplanar or a planar support. Furthermore, Brandes *et al.* [6] consider so-called path-based supports

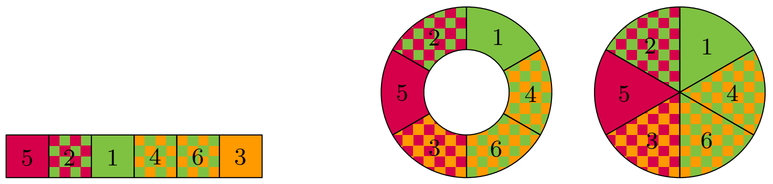

Figure 1: Subdivision drawings for $H = (V, \mathcal{S})$ with $V = \{1, \ldots, 6\}$; with $\mathcal{S} = \{(2,5), (1,2,4,6), (3,4,6)\}$ H has a path support (left) and with $\mathcal{S} = \{(2,3,5), (1,2,4,6), (3,4,6)\}$ H has a cycle support (right).

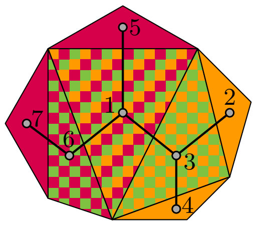

Figure 2: Subdivision drawings for $H = (V, \mathcal{S})$ with $V = \{1, \dots, 7\}$ and $\mathcal{S} = \{(1,3,6),(1,2,3,4),(1,5,6,7)\}$.

of hypergraphs where each hyperedge induces a Hamiltonian subgraph in the support. They show how to test in polynomial time whether a hypergraph has a path-based tree support.

Path or cycle supports naturally lend themselves to the creation of pleasing and easily readable subdivision drawings which are *simple* and *compact* [12] (see Fig. 1). However, not many hypergraphs admit a path or a cycle support. Tree supports, on the other hand, can have vertices of arbitrarily high degree and hence may not result in easily interpretable subdivision drawings. Therefore we consider tree supports of bounded (constant) vertex degree. For example, a binary tree support can be interpreted as the dual graph of a triangulation of a (convex) polygon and as such can be used to create a simple and compact subdivision drawing where each face of the subdivision is a triangle (see Fig. 2). Generally speaking, tree supports of constant vertex degree allow for subdivision drawings where each vertex region has constant complexity (the maximum of 3 and the vertex degree). Furthermore, each hyperedge can be drawn as a convex polygon of complexity linear in its number of vertices (generalizing the construction in Fig. 2).

Results. In Section 3 we give an $O(n^3 + kn^2)$ time constructive algorithm based on a flow formulation that solves the following decision problem: given a hypergraph H, with n vertices and k hyperedges, together with degrees d_i for each vertex i, is there a tree support for H such that the corresponding vertex i of the tree has degree at most d_i? Additionally, in Section 4 we strengthen the result by Johnson and Pollak by proving that it is even NP-complete to decide if a hypergraph has a 2-outerplanar support. Our construction also gives a much simpler proof of Johnson and Pollak's original result.

Notation and Definitions. Our input is a hypergraph $H = (V, \mathcal{S})$ with n vertices and k hyperedges. The total size of the input is $N := \sum_{i=1}^{k} |S_i|$. We interpret H as a set system $\mathcal{S} = \{S_1, \dots, S_k\}$ on a base set $V = \{1, \dots, n\}$ of n elements. Two elements h and j of V are *equivalent* with respect to $\mathcal{S}$ if every set $S_i \in \mathcal{S}$ contains either none or both of h and j. To simplify the discussion we assume that no two elements of V are equivalent. Note that we can easily construct a hypergraph with this property by replacing equivalent elements by a single element. We also assume that each element of the base set occurs in at least one set (hence $N \geq n$) and that the elements within each set are sorted. The vertices of a planar support G correspond to the elements of V. We often directly identify a vertex with "its" element and use the same name to refer to both. Furthermore, for each hypergraph $H = (V, \mathcal{S})$ we consider a graph $G(H)$ on V. Two elements u and v of V are connected by an edge in $G(H)$ if there is a hyperedge $S_i \in \mathcal{S}$ that contains both u and v. This means that every hyperedge forms a complete subgraph of $G(H)$. We define the connected components of H as the connected components of $G(H)$. Finally, a graph G is k-outerplanar if for $k = 1$, G is outerplanar and for $k > 1$, G has a planar embedding such that if all vertices on the exterior face are deleted, the connected components of the remaining graph are all $(k-1)$-outerplanar.

2 Path, Cycle, and Tree Supports

In this section we summarize previous work on path, cycle and tree supports. A graph G is a path support for a hypergraph H if G is a support and a path. Similarly, G is a cycle or tree support for H if G is a support and a cycle or tree, respectively. For all three classes of graphs one can decide whether a given hypergraph has such a support in linear time.

Path support. Korach and Stern [13] observed that the decision problem for path supports is equivalent to finding a permutation π of $\{1, \ldots, n\}$ such that, for every set S_i, the elements of S_i are consecutive in π. This problem in turn is directly related to the *consecutive ones property*: a matrix of zeroes and ones is said to have the consecutive ones property if there is a permutation of its columns such that the ones in each row appear consecutively. Let M be a matrix with n columns and m rows such that entry (i, j) is 1 if $j \in S_i$, and 0 otherwise. H has a path support if and only if M has the consecutive ones property (see Fig. 3). There are algorithms [4, 10] that can test the consecutive ones property and produce a corresponding permutation in $O(m + n + r)$ time, where $m \times n$ is the size of M, and r is the number of ones in M. Hence a path support for a given hypergraph can be found in $O(N)$ time.

Cycle support. Finding a cycle support for a hypergraph H can be reduced to finding a path support for an auxiliary hypergraph H'. For a cycle support, a set S_i is connected if and only if its complement S_i^c is connected. For some $j \in V$, let H' be the hypergraph obtained by replacing the sets S_i for which

$$\begin{array}{c} 1\,2\,3\,4\,5\,6 \\ \begin{pmatrix} 1\,0\,1\,1\,0\,1 \\ 0\,1\,0\,0\,1\,1 \\ 1\,1\,1\,0\,0\,1 \\ 0\,1\,1\,0\,0\,1 \end{pmatrix} \end{array} \begin{array}{l} \\ \{1,3,4,6\} \\ \{2,5,6\} \\ \{1,2,3,6\} \\ \{2,3,6\} \end{array} \longrightarrow \begin{array}{c} 4\,1\,3\,6\,2\,5 \\ \begin{pmatrix} 1\,1\,1\,1\,0\,0 \\ 0\,0\,0\,1\,1\,1 \\ 0\,1\,1\,1\,1\,0 \\ 0\,0\,1\,1\,1\,0 \end{pmatrix} \end{array}$$

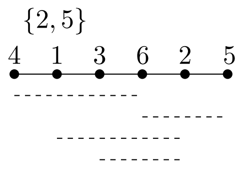

Figure 3: Finding a path support via the consecutive ones property.

$j \in S_i$ with S_i^c. As no set of H' contains j, H has a cycle support if and only if H' has a path support. By choosing j as the element that occurs in the minimum number of sets, one can reduce the problem of finding a cycle support for H to finding a path support for a hypergraph H' of size $O(N)$. This can be found in $O(N)$ time as described above. Finding a cycle support is also directly related to testing matrices for the *circular ones property* [18].

Tree support. Johnson and Pollak [11] argued that one can efficiently decide whether a hypergraph has a tree support by considering its dual. The dual of a hypergraph $H = (V, \mathcal{S})$ is the hypergraph H^*, such that each hyperedge of H corresponds to a vertex of H^*, and each vertex $v \in V$ of H corresponds to a hyperedge of H^* that contains all hyperedges of H (vertices of H^*) that contain v. The dual of a hypergraph with a tree support is an acyclic hypergraph [2], and acyclicity can be tested in linear time [17].

Korach and Stern [13] considered the following generalization of finding a tree support: assume that for a hypergraph H a real weight is given for every pair of different numbers in the vertex set V, i.e., for each potential edge in the tree. They showed that the tree support with minimum total edge weight (if it exists), can be found in polynomial time.

3 Bounded-Degree Tree Supports

We describe an algorithm that solves the following decision problem: given a hypergraph $H = (V, \mathcal{S})$ together with degrees d_i for each element i of the base set V, is there a tree support for H such that each vertex i of the tree has degree at most d_i? Our algorithm is constructive and computes a support if it exists. To simplify the discussion we assume that $V \in \mathcal{S}$. This enforces that any support is connected and does not influence the outcome of the decision problem.

To construct a bounded-degree tree support we need to know our choices when connecting vertices. Consider the sets $S_1 = \{1, 2, 3\}$ and $S_2 = \{2, 3, 4\}$, all tree supports are shown in Fig. 4. Each support has an edge connecting 2 to 3, but 1 and 4 can be connected to either 2 or 3. So it appears that the intersection $\{2, 3\}$ of S_1 and S_2 must be connected in any tree support. Korach and Stern proved this observation in [13] in the context of matroid theory, for completeness we include a simple direct proof.

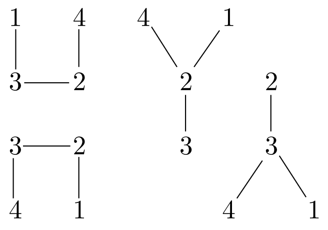

Figure 4: All tree supports.

Observation 1 *The intersection $A \cap B$ of two sets $A, B \in \mathcal{S}$ must be connected in every tree support.*

Proof: Since $A \cap B$ is always connected if it contains zero or one elements, we assume that $|A \cap B| \geq 2$. Let T be a tree support for H. So A and B are both

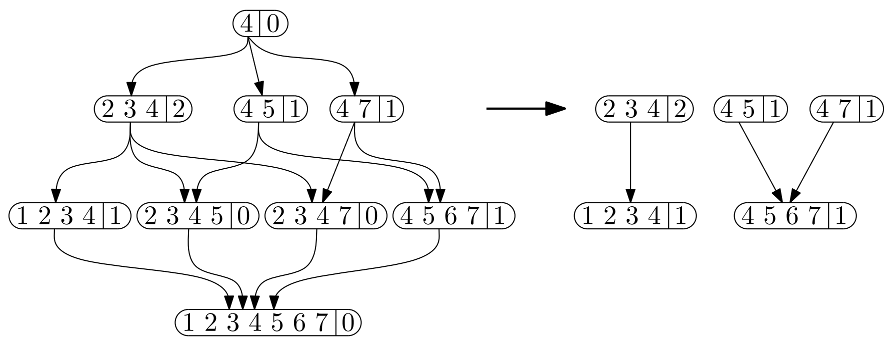

Figure 5: The intersection structure for $\{\{1,2,3,4\},\{2,3,4,5\},\{4,5,6,7\},\{2,3,4,7\},\{1,2,3,4,5,6,7\}\}$ (with the demands next to the sets) and the corresponding connectivity structure.

connected in T. Let $x \in A \cap B$ and $y \in A \cap B$. Since A is connected in T, there is a path in T from x to y using only vertices from A. Also there is a path in T from x to y using only vertices from B. Since paths in trees are unique it follows all vertices on the path from x to y are in $A \cap B$. So $A \cap B$ is connected in T. $\square$

Let $\mathcal{S}^*$ denote the set of all possible sets that can be obtained by intersecting any number of sets from $\mathcal{S}$. Clearly $\mathcal{S}^*$ is closed under intersection and $\mathcal{S} \subseteq \mathcal{S}^*$. Observation 1 implies that H has a (bounded-degree) tree support if and only if $H^* = (V, \mathcal{S}^*)$ does. We now define the *intersection structure* $\mathcal{I}$ as follows. $\mathcal{I}$ is a directed acyclic graph whose vertices are the sets in $\mathcal{S}^*$. $\mathcal{I}$ has a directed edge (S_1, S_2) if and only if $S_1 \subset S_2$ and for no set $S_3 \in \mathcal{S}^*$, we have $S_1 \subset S_3 \subset S_2$. That is, edges are directed from smaller to larger sets and represent direct containment—$\mathcal{I}$ does not contain transitive edges (see Fig. 5 (left)).

The minimum number of edges of any support of a hypergraph H can be deduced directly from its intersection structure. Let B and $A_1, \ldots, A_h$ be vertices of $\mathcal{I}$ such that (A_j, B), $1 \leq j \leq h$, are incoming edges of B in $\mathcal{I}$ and there are no further incoming edges of B. We call the sets A_j the *children* of B, and B is a *parent* of each A_j. Let us assume that the sets A_j are connected in a support G of H and that G has the fewest edges among all supports with that property. Let c be the number of connected components implied by the sets A_j, i.e., the number of connected components of the hypergraph $(B, \{A_1, \ldots, A_h\})$. To connect B we need to add at least $c-1$ additional edges to G—the *demand* of B. For example, take $B = \{4,5,6,7\}$ (and $A_1 = \{4,5\}$, $A_2 = \{4,7\}$) as shown in Figure 5 (left). Although B contains 4 elements, the sets A_1 and A_2 imply that $\{4,5,7\}$ is one connected component of B. The other connected component of B is the singleton $\{6\}$. As a result, the demand of B is $c-1=1$. The sum of the demands of all sets in $\mathcal{S}^*$ is the *total demand*.

Lemma 1 *The total demand of the sets in $\mathcal{S}^*$ equals the minimum number of edges required for any support of H.*

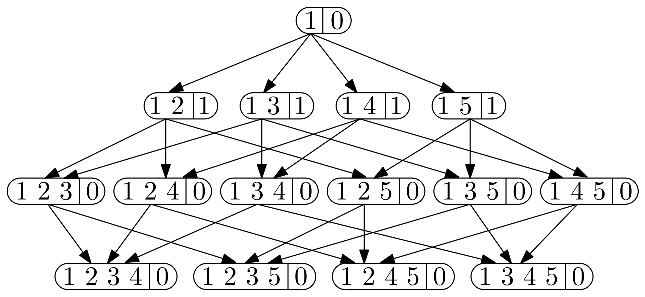

Figure 6: The intersection structure for $\{\{1,2,3,4\},\{1,2,3,5\},\{1,2,4,5\},\{1,3,4,5\}\}$.

Proof: By definition, the demand of a set B is the number of edges required to connect B, given that its children in $\mathcal{I}$ are connected. It remains to argue that no edge of a support G can simultaneously connect two sets B and B'. Assume that $|B'| \leq |B|$. The statement is obviously true if $B \cap B' = \emptyset$. If $B' \subset B$ then B' is part of a single connected component of B and hence no edge that is used to connect B connects two elements of B'. Finally, if $B \cap B' = A \neq \emptyset$, then, because $\mathcal{S}^*$ is closed under intersection, A must be a vertex of $\mathcal{I}$ as well. If an edge e of G is used to connect simultaneously both B and B', then e must connect two elements of A. But then e counts towards the demand of A. $\square$

Recall that we assume that the base set V is an element of $\mathcal{S}$. Then, by Lemma 1, a hypergraph H has a tree support if and only if the total demand equals $n-1$. $\mathcal{I}$ also indicates between which vertices the edges of a support should be. As described above, the set $\{4,5,6,7\}$ in Figure 5 has a demand of 1. Since the connected components are $\{4,5,7\}$ and $\{6\}$, the support must contain an edge between 6 and either 4, 5, or 7.

$\mathcal{I}$ contains all necessary information to answer our decision problem, but it can have exponential complexity even if H has a tree support (see Figure 6). Consider the set $\mathcal{S}$ of all but one subsets of size $n-1$ of $V = \{1, \dots n\}$. There must be one element j that is contained in each set of $\mathcal{S}$. The star graph with j as center is a tree support for $H = (V, \mathcal{S})$. However $\mathcal{S}^*$ is nearly the complete powerset of V and exponential in size. Hence we restrict ourselves to the *connectivity structure*, a limited version of the intersection structure for which we prove that it still carries all necessary information.

Connectivity structure. We say that sets with zero demand are *implied.* We remove all sets with zero demand from $\mathcal{S}^*$ and call the resulting set $\mathcal{S}^-$. The connectivity structure $\mathcal{C}$ is built on $\mathcal{S}^-$ in the same manner as the intersection structure on $\mathcal{S}^*$ (see Fig. 5 (right)). The demand of a set in $\mathcal{C}$ equals its demand in $\mathcal{I}$. If H has a tree support then $\mathcal{S}^-$ contains at most $n-1$ sets. One can easily construct examples where also in this case $\mathcal{C}$ has $\Omega(n^2)$ edges.

Clearly we do not want to compute $\mathcal{S}^-$ and the connectivity structure by first constructing $\mathcal{S}^*$ and the intersection structure and pruning sets with zero de-

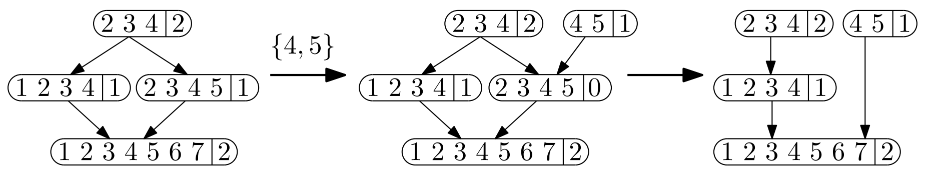

Figure 7: Incremental construction of the connectivity structure.

mand. Instead we incrementally compute a graph that is the connectivity structure if H has a tree support. Let $\mathcal{S} = \{S_1, \dots, S_k\}$ with $S_1 = V = \{1, \dots n\}$. We incrementally compute the connectivity structures $\mathcal{C}_i$ $(1 \leq i \leq k)$ for the sets $S_1, \dots, S_i$. To compute $\mathcal{C}_{i+1}$ from $\mathcal{C}_i$, we first compute all intersections between the new set S_{i+1} and all sets in $\mathcal{C}_i$. We then add those intersections which are not implied to the connectivity structure, starting with the smallest set by inclusion (see Fig. 7). If as a result any previous sets become implied, then we remove them. If at any point the total demand exceeds $n-1$, then we directly stop and conclude that the hypergraph has no tree support. We argue in the lemmas below that this approach is indeed correct.

The graph computed by this incremental construction might conceivably be missing sets since the intersection of a new set with a (removed) implied set might not be implied itself and hence should have been included. However, we can argue that for hypergraphs with a tree support this incremental approach indeed computes the correct connectivity structure (Lemma 2). But, if a hypergraph has no tree support, then the algorithm computes a total demand greater than $n-1$. Equivalently, if the total demand determined by the algorithm is $n-1$, then the hypergraph has a tree support (Lemma 3).

Lemma 2 *The incremental approach described above correctly computes the connectivity structure $\mathcal{C}$ if the hypergraph H has a tree support.*

Proof: We could use a similar approach to compute the complete intersection structure. So it remains to argue that removing implied sets in an intermediate stage does not influence the final result for hypergraphs with a tree support.

Assume that we have removed an implied set S from $\mathcal{C}_i$. Hence there must be sets $A_1, \dots, A_h$ in $\mathcal{C}_i$ that imply that S is connected. Note that $A_j \subset S$ for all $1 \leq j \leq h$. Let S' be a new set that is added to $\mathcal{C}_i$. We have to argue that $S' \cap S$ is implied if H has a tree support. In fact we show that $S' \cap S$ is implied by the sets $A'_j = S' \cap A_j$. Assume for contradiction that this is not the case and hence the sets A'_j form at least two connected components in $S' \cap S$. Because $S' \cap S$ must be connected, these connected components are directly connected by edges in a tree support. However, because the sets A_j imply the connectedness of S, these connected components are also connected in a different manner in the tree support, introducing a cycle, which contradicts the assumption that H has a tree support. Since the total demand of a hypergraph with a tree support is $n-1$, the algorithm does not terminate early for such hypergraphs. $\square$

Lemma 3 *If the total demand during the incremental construction is $n-1$ then H has a tree support.*

Proof: [by induction] $S_1 = \{1, \dots n\}$ has a demand of $n-1$ and clearly has a tree support. Now assume that the sets $S_1, \dots, S_i$ have a tree support T. In the inductive step we add the set S_{i+1} to $\mathcal{C}_i$, that is, we add the non-implied intersections of S_{i+1} with the sets in $\mathcal{C}_i$ starting with the smallest by inclusion. Let S be one of these intersections. After S has been added to $\mathcal{C}_i$, it has exactly one parent P. If it had two or more parents then it would be the non-implied intersection of at least two sets in $\mathcal{C}_i$ and as such already be contained in $\mathcal{C}_i$. If S had no parent then its demand would have to be zero for the total demand not to exceed $n-1$. Hence S would be implied.

Let P be the parent of S and let $A_1, \dots, A_h$ be the children of P before adding S. Assume that $(P, \{A_1, \dots, A_h\})$ had c connected components before S was added and that S connects x of these components into one connected component. Then the demand of P becomes $c-x$. Since the total demand remains $n-1$, the demand of S becomes $x-1$. Since all children of S are former children of P none of the demand of S can be subsumed by its children. Let $B_1, \dots, B_x$ be the connected components of $(P, \{A_1, \dots, A_h\})$ that were connected by S. We change the tree support T as follows. Disconnect the connected components of P in T. Let $B'_j = S \cap B_j$ for $1 \leq j \leq x$. All B'_j are connected in T, because these intersections have already been added. Now use the $x-1$ edges covering the demand of S to connect the B'_j into a tree. Finally we connect S with the remaining connected components of $(P, \{A_1, \dots, A_h\})$, using the $c-x$ edges covering the demand of P. By construction the new tree still connects all sets of $\mathcal{C}_i$, all intersections already added, and S. □

Lemma 3 directly implies that if H does not have a tree support then the total demand necessarily exceeds $n-1$ at some point during the construction.

Efficient algorithm. The connectivity structure can easily be computed in $O(kn^3)$ time as described above. Here we describe a more efficient algorithm following the same incremental approach. Computing $\mathcal{C}_{i+1}$ from $\mathcal{C}_i$ requires three steps: (i) add intersections between S_{i+1} and all sets in $\mathcal{C}_i$, (ii) compute the demands of all sets in the new $\mathcal{C}_i$, and (iii) remove all implied sets. To perform the third step efficiently, we actually construct the transitive closure of the connectivity structure $\mathcal{C}_i$. Note that this makes the third step trivial. We can finally obtain the correct connectivity structure $\mathcal{C}_k$ by computing the transitive reduction in $O(n^3)$ time.

For the first two steps we compute the topological order of all sets in $\mathcal{C}_i$ in $O(n^2)$ time. Following this order, we compute for each set $S \in \mathcal{C}_i$ the intersection $S' = S \cap S_{i+1}$ and add it to $\mathcal{C}_i$. Next we need to add all edges incident on S'. For the incoming edges, consider all sets $B \in \mathcal{C}_i$ such that there is an edge from B to S in $\mathcal{C}_i$ (e.g. $B \subset S$). If $B \subseteq S_{i+1}$, then add an edge from B to S', otherwise do not. If it turns out that S' is already present in $\mathcal{C}_i$, then we do not have to add it again and we can ignore it. For the outgoing edges, simply add an edge from S' to S and all sets $B \in \mathcal{C}_i$ such that there is an edge from S to B

in $\mathcal{C}_i$. Note that we can check in $O(1)$ time whether $B \subseteq S_{i+1}$ by maintaining this information for each processed set. To check if S' is already present in $\mathcal{C}_i$, note that $B = S'$ if and only if $B \subseteq S'$ and $|B| = |S'|$. Hence, by maintaining the cardinalities of each set, we can check this in $O(n)$ time.

For the second step, again follow the topological order. We maintain a graph T on $V = \{1, \ldots, n\}$ to represent the connectivity information. For a set $S \in \mathcal{C}_i$, consider the subgraph of T induced by S. The demand of S is exactly the number of connected components of this subgraph minus one. Next we arbitrarily connect these connected components in T using a number of edges equal to the demand of S. The topological order ensures that the demands are computed correctly.

Lemma 4 *The algorithm described above correctly computes the connectivity structure in $O(n^3 + kn^2)$ time if the hypergraph H has a tree support.*

Proof: For the first step we need to prove that all edges incident on S' are computed correctly. For the incoming edges, if $B \subset S'$, then $B \subset S$, so we consider all candidates. Now, if $B \subseteq S_{i+1}$, then $B \subseteq S \cap S_{i+1} = S'$. On the other hand, if $B \nsubseteq S_{i+1}$, then $B \nsubseteq S'$, as $S' \subseteq S_{i+1}$. For the outgoing edges, note that all added edges must be correct, but we might be missing some. Consider a set $B \in \mathcal{C}_i$ such that $S \nsubseteq B$ and $S' \subset B$. By the definition of the connectivity structure, $\mathcal{C}_i$ must contain $S'' = S \cap B$, unless S'' is implied. If S'' is not implied, then note that $S' = S' \cap B = (S \cap S_{i+1}) \cap B = (S \cap B) \cap S_{i+1} = S'' \cap S_{i+1}$. As S'' comes before S in the topological order, this means that S' was already present in $\mathcal{C}_i$. If S'' is implied, then there must be sets $A_1, \ldots, A_h$ in $\mathcal{C}_i$ that imply that S'' is connected. Note that the sets $A'_j = A_j \cap S_{i+1}$ $(1 \leq j \leq h)$ must already have been added to $\mathcal{C}_i$. Using the same arguments as in the proof of Lemma 2, we can conclude that the sets A'_j imply that $S'' \cap S_{i+1}$ is connected if H has a tree support. But, as shown above, $S' = S'' \cap S_{i+1}$. So S' is implied and will be removed in the third step. This means that its outgoing edges need not be correct.

The correctness of the second step is already argued above. All that remains is the running time. For the first two steps, every set can be processed in $O(n)$ time. So every step can be computed in $O(n^2)$ time (including the topological sorting), which means that we can compute $\mathcal{C}_{i+1}$ from $\mathcal{C}_i$ in $O(n^2)$ time. Finally we compute the transitive reduction in $O(n^3)$ time, which results in a total running time of $O(n^3 + kn^2)$. □

If the hypergraph H does not have a tree support, the efficient algorithm will also notice this by an increase in demand. This does not change the running time of the algorithm.

Flow formulation. Using the connectivity structure $\mathcal{C}$ we can formulate our decision problem as a flow problem. To simplify matters we add some additional sets to $\mathcal{C}$. Let S be a vertex of $\mathcal{C}$ and let $A_1, \ldots, A_h$ be children of S such that $A_1, \ldots, A_h$ form a (maximal) single connected component C of S. We say that C is a *connection set* (or *c-set* for short) and add C to $\mathcal{C}$ in between $A_1, \ldots, A_h$

and S. By construction all c-sets have zero demand. We also add all singleton sets. The resulting graph $\mathcal{C}^*$ is called the *augmented connectivity structure.* Every set in $\mathcal{C}^*$ is either a singleton set, a c-set, or a *normal* set. Normal sets now have the property that all their children are disjoint, hence the demand of a normal set is the number of its children minus one. Let c_S be the number of connected components of a set S in $\mathcal{C}$. The number of c-sets we add to $\mathcal{C}^*$ is $k_c \leq \sum_S c_S \leq (n-1) + \sum_S (c_S - 1) = 2n-2$. So $\mathcal{C}^*$ has $O(n)$ vertices as well.

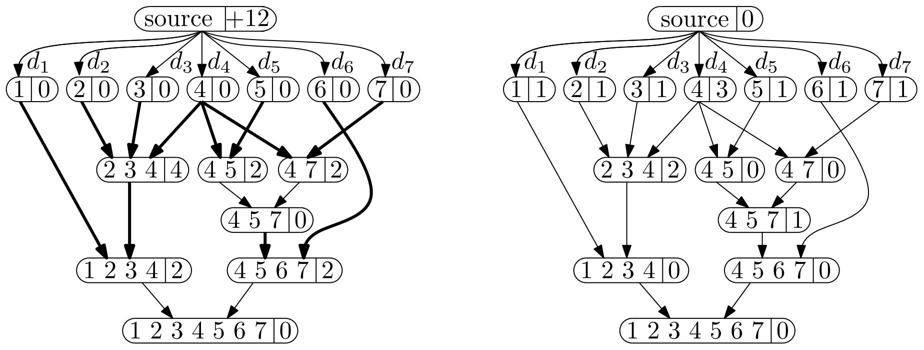

Figure 8: The original (left) and the new (right) flow network. Thick edges denote a lower bound on the flow. In the flow networks the production/consumption (left) or the capacity to the sink (right) is shown instead of the demand.

We construct a flow network $\mathcal{F}$ from $\mathcal{C}^*$ as follows. We add a source and connect it to the singletons with edges whose capacities are the maximal degree of each element; the edge from the source to $\{i\}$ has capacity d_i. The capacities of the remaining edges are unbounded. Every incoming edge of a normal set requires at least one unit of flow, so we have a lower bound for the flow on these edges. The source produces $2n-2$ units of flow which is consumed by the normal sets, each normal set consumes twice its demand (see Fig. 8 (left)). Intuitively the units of flow correspond to the degrees of the vertices in the tree support. Consider a normal set S and its children $A_1, \ldots, A_h$. Since these children are disjoint in $\mathcal{C}^*$ we need at least one unit of flow from each A_i to connect S. Also, S has to consume exactly $2h-2$ units of flow.

Observation 2 (Tamura and Tamura [16]) *For a given degree sequence* $(d_1, \ldots, d_h)$ *with* $d_i \geq 1$ *for all* i*, a tree exists whose vertices have precisely these degrees if and only if* $\sum_{j=1}^{h} d_j = 2h-2$.

Lemma 5 *Every tree support* T *that respects the degree bounds corresponds to a feasible flow* F.

Proof: As argued in the proof of Lemma 1 each edge $e = \{u, v\}$ of T can be mapped uniquely to a normal set S of $\mathcal{C}^*$. Let $A_1, \ldots, A_h$ be the children of S. We have $u \in A_i$ and $v \in A_j$ for some $i \neq j$, $1 \leq i, j \leq h$. We choose an arbitrary path from the source to S through $\{u\}$ and A_i and add a unit of flow to every edge on this path. We do the same for $\{v\}$ and A_j. Repeating this procedure for every edge of T constructs a flow F. It remains to argue that F is feasible. Consider again a normal set S and one of its children A_i. Since S is connected in T there is at least one edge of T which is mapped to S and contains an element of A_i. Hence the edge from A_i to S has at least one unit of flow. By the fact that $h-1$ edges are mapped to S, the number of paths from the source to S is exactly $2h-2$, so S consumes the correct amount of flow. Finally, we add flow exactly once for each edge in T and so the flow from the source to a singleton $\{i\}$ is at most d_i. □

Before we explain how to construct a valid tree support from a feasible flow, we first discuss how to compute such a feasible flow using a standard construction (see [1], Sections 6.2 and 6.7): we transform the flow network $\mathcal{F}$ to a max-flow network $\mathcal{F}'$. We remove the lower bounds and the production/consumption restrictions. We add a sink to $\mathcal{F}$ and add edges from all sets to this sink. For a set S let δ_S denote its demand. As capacity of the edge from a set S to the sink we take the number of outgoing edges to a normal set (edges that had a lower bound of one), and if S is a normal set, we further add $\delta_S - 1$ to the capacity (Fig. 8 (right)). The sum of the capacities of the incoming edges of the sink is now exactly $2n-2$. The following is shown in [1].

Lemma 6 *A feasible flow exists for $\mathcal{F}$ if and only if the max-flow of $\mathcal{F}'$ is exactly $2n-2$.*

Tree construction. We now describe how to construct a tree support T from a feasible flow for $\mathcal{F}$. Although the flow tells us the degrees for each vertex in T, we need to use the entire flow to build T correctly. We handle the sets of $\mathcal{F}$ in topological order. Consider a normal set S with children $A_1, \ldots, A_h$. We can use the flow consumed by S to connect the sets A_j in T. To know which vertices need to be connected in T, we need to know from which singletons the flow to a set A_j originates. We maintain this information using a list L_i for each set S_i. The list L_i contains a vertex number for every unit of outgoing flow. Initially the list L_j for a singleton $\{j\}$ contains a number of copies of j corresponding to the amount of outgoing flow. Now let S again be a normal set S with children $A_1, \ldots, A_h$. First we build a tree on the components A_j depending on the flow towards S (note that the ingoing flow of S might exceed $2h-2$, but in that case

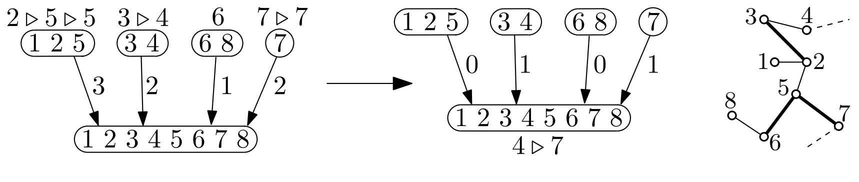

Figure 9: A step in the tree construction algorithm. The lists are shown above/below the sets. The thick edges have been added to the tree support.

we can make choices as long as we use one unit of flow from each child). If we want to connect A_i to A_j, then we simply take the first elements x of L_i and y of L_j and add an edge (x, y) to T (Fig. 9). Then we remove x and y from L_i and L_j, respectively. After the tree is built for S, we take k_j elements from each list L_j of A_j, where k_j is the remaining flow from A_j to S, and merge them into the list L of S. In case S is a c-set, we perform only this final step.

Lemma 7 *The method described above correctly constructs a tree support T, which respects the degree bounds, from a feasible flow.*

Proof: We have to show three things: (i) every set S_i is connected in T, (ii) T is a tree and (iii) the degree bounds are respected. The algorithm adds exactly $n-1$ edges to T, so (ii) follows from (i) since V is an element of $\mathcal{S}$ and hence T is necessarily connected. When we handle a set S_i, we make sure that it is connected in T. Since we never remove edges from T, (i) holds. Finally, when we add an edge incident to a vertex x to T, we remove it from a list. Note that vertex numbers are added to the singleton lists, but after that they are only moved from list to list. So the degree of x can be at most the size of L_x, which is properly bounded in a feasible flow. $\square$

Theorem 1 *Given a hypergraph H, with n elements and k sets, together with degrees d_i for each element i of the base set, we can construct a tree support T for H such that each vertex i of T has degree at most d_i in $O(n^3+kn^2)$ time—if such a tree support exists.*

Proof: The first step is to compute the connectivity structure. By Lemma 4, this can be computed in $O(n^3+kn^2)$ time. Then we can augment the connectivity structure and construct the max-flow network in $O(n^3)$ time. We compute the max-flow using the Ford-Fulkerson algorithm [8], which runs in $O(|E|f^*)$ time, where $|E|$ is the number of edges in the flow network and f^* is the maximum flow. As f^* is $O(n)$, this takes at most $O(n^3)$ time. Finally we construct the tree in $O(n^2)$ time. $\square$

4 Hardness Results

We first show that for any instance of 3-SAT (or of SAT), we can reduce it to an instance of finding a planar support for a hypergraph such that there is a planar support if and only if the 3-SAT instance is satisfiable. The planar support—if one exists—will be 3-outerplanar (or we can assume it to be so without limiting any options in the planar support). We then modify our construction to reduce 3-SAT (or SAT) to finding a 2-outerplanar support for a hypergraph. In the modified construction, if a 3-SAT instance is not satisfiable, the corresponding hypergraph might still have a planar support.

The first part of the construction is the same for the reduction to planar and to 2-outerplanar supports. Let $a, b, c, \ldots, y$ be the variables used in a 3-SAT instance. We represent each variable, say b, by six elements $b_1, \ldots, b_6$

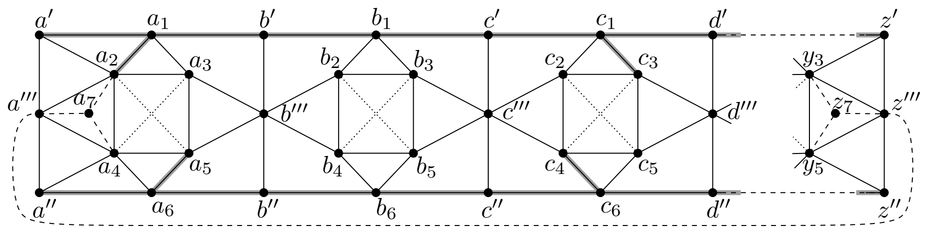

Figure 10: Construction of a hypergraph and its planar/2-outerplanar support from a 3-SAT instance.

and some sets. Many of these sets have size two, and so any planar support must include an edge that connects the vertices of these two elements. The sets for b are $\{b_1, b_2\}$, $\{b_2, b_3\}$, $\{b_1, b_3\}$, $\{b_2, b_4\}$, $\{b_3, b_5\}$, $\{b_4, b_5\}$, $\{b_5, b_6\}$, and $\{b_4, b_6\}$, see Fig. 10. We connect the variable elements into some sequence (in any order; we assume it is $a, b, c, \ldots$) by extra elements $a', b', c', \ldots, y', z'$ and $a'', b'', c'', \ldots, y'', z''$, and use sets $\{a', a_1\}$, $\{a_1, b'\}$, $\{b', b_1\}$, etc., and $\{a'', a_6\}$, $\{a_6, b''\}$, $\{b'', b_6\}$, etc. We also use extra elements $a''', b''', c''', \ldots, y''', z'''$ and sets $\{a', a'''\}$, $\{a''', a''\}$, $\{b', b'''\}$, $\{b''', b''\}$, etc., to separate the variables from each other. Next, we use sets $\{a''', a_2\}$, $\{a''', a_4\}$, $\{b''', a_3\}$, and $\{b''', a_5\}$ for each variable, and four more sets $\{a', a_2\}$, $\{a'', a_4\}$, $\{z', y_3\}$, and $\{z'', y_5\}$. All sets of cardinality two imply a 3-connected planar graph as a support, so its embedding is fixed up to the choice of the outer face.

For the reduction to planar supports, we add one more set, namely $\{a''', z'''\}$ (see Fig. 10), which ensures that no edge between any of $a', a_1, b', b_1, c', c_1, \ldots$ and any of $a'', a_6, b'', b_6, c'', c_6, \ldots$ can exist in the planar support.

A 3-SAT clause $(a \vee \overline{c} \vee x)$ is represented by a set

$$\{a_1, b_1, c_1, \ldots, a', b', c', \ldots, a_6, b_6, c_6, \ldots, a'', b'', c'', \ldots, a_2, a_5, c_3, c_4, x_2, x_5\} .$$

In Fig. 10, these are all vertices of the top row, all vertices of the bottom row, the subscript-2 and subscript-5 vertices of the variables that occur as a literal in the clause, and the subscript-3 and subscript-4 vertices of the variables that occur negated as a literal in the clause. Their connection in the fixed part of the planar support is shown in grey in the figure.

The only way to extend the fixed part of the planar support so that the set of a clause has a connected support is to use at least one of the edges (a_2, a_5), (c_3, c_4), or (x_2, x_5). The only choices of edges in the support that can help to give sets planar support are ones like (a_2, a_5) and (a_3, a_4) (dotted in Fig. 10).

For any variable, it is easy to see that we can only take the edge (a_2, a_5) or (a_3, a_4), and not both, otherwise the support graph is not planar. This corresponds to the variable assignment of a to TRUE (take edge (a_2, a_5)) or FALSE (take edge (a_3, a_4)). Hence, the 3-SAT instance has a variable assignment that makes it true if and only if the constructed hypergraph has a planar support. It is easy to see that the planar support is 3-outerplanar. Hence, if a planar support exists, then a 3-outerplanar support exists.

Theorem 2 *It is NP-complete to decide whether a hypergraph has a 3-outer-planar support.*

Next, we modify our construction to show that it is NP-hard to decide whether a hypergraph has a 2-outerplanar support. Instead of the set $\{a''', z'''\}$ we add the elements a_7 and z_7 and the sets $\{a''', a_7\}$, $\{a_2, a_7\}$, $\{a_4, a_7\}$, $\{z''', z_7\}$, $\{y_3, z_7\}$, and $\{y_5, z_7\}$.

By the previous arguments, if an instance of 3-SAT is satisfiable then the corresponding hypergraph has a 2-outerplanar support. Now, if an instance of 3-SAT is not satisfiable then the hypergraph has a planar support (with an edge between one of the vertices $a', a_1, \dots z'$ of the top row and one of the vertices $a'', a_6, \dots, z''$ of the bottom row), but it is not 2-outerplanar: there is no face of the support such that removing the vertices of this face would make the support outerplanar.

Theorem 3 *It is NP-complete to decide whether a hypergraph has a 2-outer-planar support.*

5 Conclusions

We described a constructive algorithm which tests in polynomial time if a given hypergraph has a planar support that is a tree where the maximal degree of each vertex is bounded. Furthermore, we strengthened a result by Johnson and Pollak by proving that it is NP-complete to decide if a hypergraph has a 2-outerplanar support. It remains open whether a polynomial-time algorithm exists to decide if a given hypergraph has an outerplanar support.

Acknowledgements

We thank the anonymous referees for pointing us to some related work. K. Buchin, B. Speckmann, and K. Verbeek were supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 639.022.707.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, 1993.

[2] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30:479–513, 1983.

[3] C. Berge. *Graphs and Hypergraphs.* North-Holland, 1973.

[4] K. Booth and G. Lueker. Testing for the consecutive ones property, interval graphs, and planarity using pq-tree algorithms. *Journal of Computer and System Sciences*, 13:335–379, 1976.

[5] U. Brandes, S. Cornelsen, B. Pampel, and A. Sallaberry. Blocks of hypergraphs - applied to hypergraphs and outerplanarity. In *Proc. 21st International Workshop on Combinatorial Algorithms (IWOCA 2010)*, volume 6460 of *LNCS*, pages 201–211. Springer, 2011.

[6] U. Brandes, S. Cornelsen, B. Pampel, and A. Sallaberry. Path-based supports for hypergraphs. In *Proc. 21st International Workshop on Combinatorial Algorithms (IWOCA 2010)*, volume 6460 of *LNCS*, pages 20–33. Springer, 2011.

[7] M. Brinkmeier, J. Werner, and S. Recknagel. Communities in graphs and hypergraphs. In *16th ACM Conference on Information and Knowledge Management*, pages 869–872, 2007.

[8] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 2nd edition, 2001.

[9] J. Flower and J. Howse. Generating Euler diagrams. In *Proc. Diagrams 2002*, volume 2317 of *LNCS*, pages 61–75. Springer, 2002.

[10] W. L. Hsu. A simple test for the consecutive ones property. *Journal of Algorithms*, 43(1):1–16, 2002.

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

[12] M. Kaufmann, M. van Kreveld, and B. Speckmann. Subdivision drawings of hypergraphs. In *Proc. 16th International Symposium on Graph Drawing (GD 08)*, volume 5417 of *LNCS*, pages 396–407. Springer, 2009.

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

[14] J. R. Lundgren. Food webs, competition graphs, competition-common enemy graphs and niche graphs. *Applications of Combinatorics and Graph Theory to the Biological and Social Sciences*, 17:221–243, 1989.

[15] G. Sander. Layout of directed hypergraphs with orthogonal hyperedges. In *Proc. 12th International Symposium on Graph Drawing (GD 04)*, volume 3393 of *LNCS*, pages 381–386. Springer, 2005.

[16] A. Tamura and Y. Tamura. Degree constrained tree embedding into points in the plane. *Information Processing Letters*, 44:211–214, 1992.

[17] 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*, 13:566–579, 1984.

[18] A. Tucker. Matrix characterizations of circular-arc graphs. *Pacific Journal of Mathematics*, 39(2):535–545, 1971.