Minimum Tree Supports for Hypergraphs and Low-Concurrency Euler Diagrams

Boris Klemz, Tamara Mchedlidze, and Martin Nöllenburg

Institute of Theoretical Informatics, Karlsruhe Institute of Technology (KIT), Germany

Abstract. In this paper we present an $O(n^2(m + \log n))$-time algorithm for computing a minimum-weight tree support (if one exists) of a hypergraph $H = (V, S)$ with n vertices and m hyperedges. This improves the previously best known algorithm with running time $O(n^4 m^2)$. A support of H is a graph G on V such that each hyperedge in S induces a connected subgraph in G. If G is a tree, it is called a tree support and it is a minimum tree support if its edge weight is minimum for a given edge weight function. Tree supports of hypergraphs have several applications, from social network analysis and network design problems to the visualization of hypergraphs and Euler diagrams. We show in particular how a minimum-weight tree support can be used to generate an area-proportional Euler diagram that satisfies typical well-formedness conditions and additionally minimizes the number of concurrent curves of the set boundaries in the Euler diagram.

1 Introduction

A hypergraph $H = (V, S)$ is a generalization of a graph that consists of a set of vertices V and a set of hyperedges S, which are arbitrary non-empty subsets of V (in contrast to graph edges, which are defined as pairs of vertices). Thus S is a subset of the power set $\mathcal{P}(V) = 2^V$. A graph $G = (V, E)$ is called a *support* of a hypergraph $H = (V, S)$ on the same vertex set if every hyperedge $s \in S$ induces a connected subgraph in G.

Sparse support graphs are interesting from the perspective of network design as they represent realizations of hypergraphs as graphs, in which the vertices of each hyperedge induce a connected component. Korach and Stern [16] introduced the problem of finding a *minimum(-weight) tree support* (MTS) for a given hypergraph $H = (V, S)$ and a given edge-weight function $w\colon \binom{V}{2} \to \mathbb{R}$ for the support graph. Here, an MTS is a support $T = (V, E)$ that is a tree with minimum total edge weight $\sum_{e \in E} w(e)$. Not every hypergraph has a tree support, but the decision problem can be solved in linear time by testing whether its dual hypergraph is acyclic [14]. If a hypergraph has a tree support, it is called a *tree-hypergraph*. Korach and Stern [16] gave an algorithm to compute an MTS in $O(|V|^4|S|^2)$ time (if it exists). They later presented another polynomial-time algorithm for a restricted variation, in which they ask for a tree support of minimum weight such that each subtree induced by a hyperedge is a star [17].

Hypergraphs and hypergraph supports are not as frequently used and studied as graphs themselves, but they still have many real-world applications. For example, in social network analysis, minimum tree supports are used to compute maximum-likelihood social networks that serve as models to explain a collection of observed disease outbreaks that are modeled as hyperedges of the infected persons [1,2]. In topic-based peer-to-peer publish/subscribe systems [7, 13] the input is a set of users V and a set of topics S, where each topic $t \in S$ is a subset of users (i.e., a hyperedge) interested in the topic. The task is to design an overlay network G on V (called *minimum topic-connected overlay*) with the minimum number of edges so that each topic forms a connected subgraph in G thus enabling private communication within each topic. If the underlying hypergraph $H = (V, S)$ admits a tree support then this minimum overlay network will be a tree; if establishing edges in G is linked with a cost, the task is again to find an MTS. The unweighted problem is also known as *subset interconnection design*, which generally asks for a support graph with the minimum number of edges, i.e., not necessarily a tree. It has applications in the design of reconfigurable networks, e.g., vacuum systems, in which valves correspond to edges in the support and their number needs to be minimized [6, 10, 11].

Of particular interest for hypergraph visualizations are *planar supports*. Johnson and Pollak [14] showed that a hypergraph is *vertex-planar* if and only if it has a planar support. A vertex-planar hypergraph has a representation of the vertices as faces in a planar subdivision such that for each hyperedge the union of the faces corresponding to the vertices incident to that hyperedge is a connected region. Simple and compact subdivision drawings [15] are an interesting restriction of vertex-planar hypergraph representations that puts additional constraint on the geometry of hyperedge representations. Johnson and Pollak [14] proved, however, that deciding the existence of a planar support is NP-complete; Buchin et al. [4, 5] extended the NP-completeness to testing the existence of a 2-outerplanar support. On the other hand, it can be decided in polynomial time, whether a hypergraph has a path-, cycle-, tree-, or cactus-support [3, 14, 16].

Contributions. In this paper we study minimum tree supports from a perspective that was initially motivated by generating (area-proportional) Euler diagrams with low concurrency of set contours. Euler diagrams are set visualizations and thus closely related to hypergraph visualizations. Section 2 describes the background of Euler diagrams, defines our algorithmic problem in that context, and sketches a solution approach based on minimum tree supports. In Section 3 we introduce some necessary definitions and notations, before we present our main technical contribution in Section 4. Our result is initially tailored for the problem to generate low-concurrency Euler diagrams. Hence we first transform an abstract Euler diagram description D into a so-called *labeled hypergraph* $H(D)$ and then present an algorithm that computes an MTS for $H(D)$ in $O(n^2 m)$ time, where n is the number of vertices and m is the number of hyperedges. The algorithm itself is a simple modification of Kruskal's algorithm for incrementally constructing minimum spanning trees, but its correctness proof relies on several crucial properties of tree supports and the special order in which the algorithm adds edges to the growing tree support. Finally, in Section 5 we generalize our result and show that every hypergraph can easily be translated into an equivalent labeled hypergraph. Then we can apply our algorithm for labeled hypergraphs to compute minimum tree supports for arbitrary hypergraphs that admit a tree support. This improves the result of Korach and Stern [16], who gave an algorithm with running time $O(n^4 m^2)$, to an algorithm with time complexity $O(n^2(m + \log n))$.

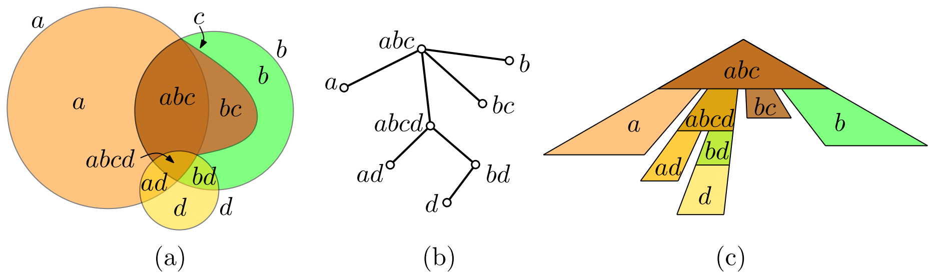

Fig. 1: (a) Euler diagram realizing AEDD $D = (\{a, b, c, d\}, \{a, b, d, ad, bd, bc, abc, abcd\})$, (b) tree support T for $H(D)$, (c) area-proportional Euler diagram based on T

2 Euler Diagrams

An *Euler diagram* $\mathcal{D}$ is a visualization of a set system as a collection of simple closed curves, whose interiors represent the sets, see Fig. 1(a) for an example. The arrangement of curves forms a subdivision of the plane and each face is called a (concrete) *zone* of the Euler diagram. We define an *abstract Euler diagram description* (AEDD) as a pair $D = (L, Z)$, where L is a set of *labels* (each representing one set in a set system) and $Z \subseteq \mathcal{P}(L)$ is a set of label subsets that we call *zones* (each representing a non-empty intersection of a particular set of labels). We say that an Euler diagram $\mathcal{D}$ *realizes* an AEDD D if there is a bijection φ between L and the set of curves in $\mathcal{D}$, as well as between Z and the set of concrete zones of $\mathcal{D}$ such that for each zone $z \in Z$ the concrete zone $\varphi(z)$ is in the interior of curve $\varphi(l)$ for each $l \in z$ and exterior to $\varphi(l')$ for each $l' \in L \setminus z$. AEDDs are closely related to hypergraphs since we can interpret each zone as a vertex and each label l as a hyperedge containing all zones that carry the label l. Similarly, an Euler diagram that realizes a given AEDD is a subdivision drawing of its corresponding hypergraph.

Euler diagrams must adhere to certain *well-formedness conditions* [8,12] that control the visual appearance of the diagrams, e.g., a zone with a certain set of labels L' may exist if and only if there exists an element that is contained in all sets corresponding to L' and that is not contained in any other set. Moreover, every zone must be uniquely labeled, i.e., there cannot be two distinct zones that lie in the interior of exactly the same set of curves. Other common well-formedness criteria require *convex zones*, disallow *triple points*, i.e., points that lie in the intersection of three or more curves, or disallow *concurrent curves* that run partially in parallel, i.e., connected intersections of two distinct curves c and c' that contain more than just one point. There is a number of algorithms and complexity results for generating Euler diagrams with certain well-formedness properties [8,12,18]. Since one can easily come up with AEDDs that require concurrencies (see the example in Fig. 1), it is an interesting problem to generally allow concurrencies, but minimize their total number in the diagram.

Another interesting variation of Euler diagrams are *area-proportional* Euler diagrams. Given an AEDD $D = (L, Z)$ together with a weight function $A\colon Z \to \mathbb{R}^+$ on the zones, the task is to find an area-proportional Euler diagram that realizes D such that the area of each concrete zone with label set $z \in Z$ is $A(z)$. Some algorithms are known for generating area-proportional Euler diagrams [9, 19], but they work heuristically or apply to very restricted inputs only.

Algorithm for tree-based area-proportional Euler diagrams. We now sketch an algorithm that generates for a given AEDD and a tree support of its induced hypergraph an area-proportional Euler diagram with convex zones. If the tree support is an MTS with respect to a specific weight function measuring internal concurrencies between neighboring zones, then the Euler diagram realizes this minimum number of internal concurrencies.

Let $D = (L, Z)$ be an AEDD. We define the *labeled hypergraph* $H(D) = (Z, S(L))$ as the hypergraph that contains a vertex for each zone of D and a hyperedge for each label of D. Each vertex $z \in Z$ is associated with the set of labels of its underlying zone. In the following we use the abbreviated notation $z = abc$ for $z = \{a, b, c\}$ that simply concatenates all labels in the zone. For each label $l \in L$ we create the hyperedge $s(l) = \{z \in Z \mid l \in z\}$, which defines the hyperedge set $S(L) = \{s(l) \mid l \in L\}$. In Section 4 we describe an algorithm that computes a minimum tree support T for a labeled hypergraph $H(D)$ and an arbitrary edge weight function $w\colon \binom{V}{2} \to \mathbb{R}$ (assuming that $H(D)$ admits a tree support). For our purposes we define w as the *concurrency* function of the AEDD D, i.e., we set $w(z, z') = |(z \cup z') \setminus (z \cap z')|$. This function counts the number of concurrent curves that run between zones z and z' if they will be selected as neighboring faces in the Euler diagram.

Now let's assume that we are given an AEDD $D = (L, Z)$ and an MTS T for its labeled hypergraph $H(D)$ provided with the concurrency function. We construct an area-proportional Euler diagram as follows (see Fig. 1). Let r be an arbitrary root of T and create a convex polygon of area $A(r)$, e.g., a triangle. Let $z_1, \ldots, z_t$ be the children of r and choose one edge σ of the root polygon. We create disjoint subsegments of σ and disjoint wedges based on these subsegments, each of which is reserved for the zones in the t subtrees of r. For each z_i we create a trapezoid of area $A(z_i)$ at the base of the i-th wedge. Then we recurse using the respective sides opposite to σ as the new base edges in the construction. It is clear that this produces convex, area-proportional faces. Since T is a support, the union of the zones of each label is connected. Moreover, since we used the concurrency function to minimize the weight of T, we have minimized the total number of concurrencies of curves running between adjacent zones.

3 Preliminaries

Let $D = (L, Z)$ be an abstract Euler diagram description, where $|L| = \lambda$. Recall that the labeled hypergraph for D is denoted by $H(D) = (Z, S(L))$. If $L' \subseteq L$ is a subset of labels, we denote the corresponding hyperedge in $H(D)$ as $S(L') = \{s(\ell) \mid \ell \in L'\}$.

In order to construct a tree support for a hypergraph we define the so-called *skeleton* $G = (Z, E)$ of $H(D) = (Z, S(L))$, which is defined as a complete weighted graph on vertex set Z, where each edge $e = \{u, v\} \in E$ is associated with the *cardinality* $c(e) = |u \cap v|$. Each tree support for $H(D)$ is a spanning subtree of the skeleton G. Since λ is the number of distinct labels in L, the maximum cardinality of an edge of G is $\lambda - 1$. We denote by E_i the set of all edges of G with cardinality i and we set $E_{\geq i} = \bigcup_{j=i}^{\lambda-1} E_j$. For a path P in G we define the *cardinality* of P as the smallest

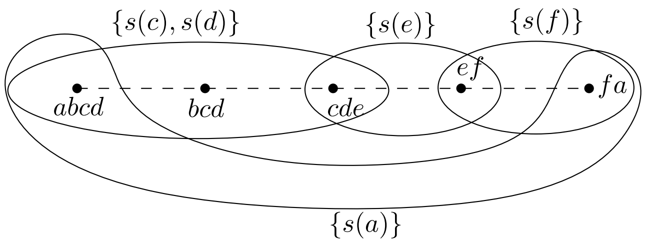

Fig. 2: A path P in G with positive cardinality and a sequence of hyperedge sets $\{s(c), s(d)\}, \{s(e)\}, \{s(f)\}, \{s(a)\}$ defined by P. This sequence forms a cycle on vertices $abcd$, cde, ef, and fa.

cardinality of its edges. For a tree T we denote by $p(u, v, T)$ the unique path connecting vertices u and v in T.

Recall that we want to compute an MTS with respect to some edge weight function $w\colon \binom{Z}{2} \to \mathbb{R}$, e.g., the concurrency function defined in Section 2. We use w as an edge weight function of the skeleton G.

For a hypergraph $H = (V, S)$ we say that an edge $\{u, v\}$ of its skeleton G *supports* a hyperedge $s \in S$, if both $u, v \in s$. An edge $\{u, v\}$ *supports* a set of hyperedges $S' \subseteq S$ if it supports all the hyperedges of S'. Let T be a spanning tree of G and P be a path in T. Path P is called *supporting* if for each hyperedge $s \in S$, the set $s \cap P$ is either empty or contains a set of consecutive vertices of P. Otherwise, P is called *non-supporting*. A non-supporting path of T is *minimal* if each sub-path of P is supporting. By recalling the definition of a tree support, we observe that a spanning tree T of G is a tree support for $H(D)$ if and only if each path of T is supporting. From this fact and the definition of the cardinality of a path in G we derive the following.

Property 1. Let $e = \{u, v\}$ be an edge in G and let T be a tree support for $H(D)$. Any edge of the path $p(u, v, T)$ supports the set $S(u \cap v)$, i.e., the path $p(u, v, T)$ has cardinality at least $c(e)$.

Let $S_1, \ldots, S_k \subseteq S(L)$ be a sequence of hyperedge sets and let $z_1, \ldots, z_k \in Z$ be a sequence of vertices of $H(D)$ such that $z_i \in s,\ \forall s \in S_i \cup S_{i+1},\ i = 1, \ldots, k-1$ and $z_k \in s,\ \forall s \in S_k \cup S_1$. Then we say that the sequence $S_1, \ldots, S_k$ forms a *cycle* on vertices $z_1, \ldots, z_k$. In Fig. 2 the hyperedge sets $\{s(c), s(d)\}, \{s(e)\}, \{s(f)\}, \{s(a)\}$ form a cycle on vertices $abcd, cde, ef, fa$.

A path P in G with non-zero cardinality *defines* a sequence of hyperedge sets as follows (see Fig. 2). Two consecutive vertices of P belong to at least one common hyperedge, since the cardinality of the edge is greater than zero. Include into S_1 those hyperedges that contain the longest initial part of P. Remove all edges of P that are in S_1 and continue recursively. Notice that if the end-vertices of P belong to a common hyperedge, which does not contain at least one internal vertex of P, then there exists a non-trivial sequence of hyperedge sets that forms a cycle on a certain subset of vertices of P.

The following lemma states a property of a path in a tree support that contains a cycle of hyperedge sets.

Lemma 1. *Let $H = (V, S)$ be a tree-hypergraph such that the sequence of hyperedge sets $S_1, \dots, S_k \subseteq S$ forms a cycle on vertices $v_1, \dots, v_k \in V$. Let $T = (V, E)$ be a tree support for H. Then, for any two distinct vertices v_i, v_j, $1 \leq i, j \leq k$, every edge of the path $p(v_i, v_j, T)$ supports at least two of the hyperedge sets $S_1, \dots, S_k$.*

Proof. Let t be an index $i \leq t < j$ of a vertex in the cycle. By the definition of a cycle formed by the sequence of hyperedge sets, $v_t \in s$, $\forall s \in S_t \cup S_{t+1}$ and $v_{t+1} \in s$, $\forall s \in S_{t+1} \cup S_{t+2}$. Therefore, both end-vertices of the subpath $p(v_t, v_{t+1}, T)$ belong to each $s \in S_{t+1}$, and therefore each edge of $p(v_t, v_{t+1}, T)$ supports S_{t+1}. (Note that all index computations are performed modulo k.)

Since T is a tree, the removal of any edge of $p(v_t, v_{t+1}, T)$ from T produces two subtrees $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$, such that $V_1 \cap V_2 = \emptyset$ and the cycle vertices $v_i, \dots, v_t \in V_1$ and $v_{t+1}, \dots, v_j \in V_2$. Let $a \geq j$ or $a < i$ be the index such that $v_a \in V_2$ and $v_{a+1} \in V_1$. The cycle among hyperedge sets $S_1, \dots, S_k$ implies that $v_a, v_{a+1} \in s$ for every hyperedge $s \in S_{a+1}$. Since T is a tree support, every edge of the path $p(v_a, v_{a+1}, T)$ supports S_{a+1}. Thus, every edge of $p(v_t, v_{t+1}, T)$ supports S_{a+1}. We conclude the proof by observing that indices $a + 1$ and $t + 1$ are distinct, since $v_{t+1} \in V_2$ and $v_{a+1} \in V_1$. □

The following lemmata are used as tools in the following section.

Lemma 2. *Let $G = (V, E)$ be a connected graph, let $T = (V, E_0 \cup E_1 \cup \dots \cup E_t)$ be a spanning-tree of G with $E_i \cap E_j = \emptyset$ for every $0 \leq i \neq j \leq t$ and let the subgraph $T_k = (V_k, E_k)$, $1 \leq k \leq t$, of T induced by E_k be connected. For any forest consisting of trees $T'_1 = (V_1, E'_1), \dots, T'_t = (V_t, E'_t)$ the graph $T' = (V, E_0 \cup E'_1 \cup \dots \cup E'_t)$ is a spanning-tree of G.*

Proof. We show that after substitution of the edges of T_1 by the edges of T'_1, the resulting graph $\tilde{T} = (V, E_0 \cup E'_1 \cup E_2 \dots \cup E_t)$ is a spanning tree of G. The result then follows by applying this procedure to the remaining $T_2, \dots, T_t$. It is trivial to see that $\tilde{T}$ is a spanning connected subgraph of G. Assume for the sake of contradiction that $\tilde{T}$ is not a tree, i.e., it contains a cycle. If we substitute the maximal paths of this cycle that belong to T'_1 by paths in T_1, we obtain a (not necessarily simple) cycle in T, which is a contradiction. □

Lemma 3. *Any tree support of $H(D)$ contains the edge set $E_{\lambda-1}$ as a subset.*

Proof. Notice that the statement is trivially true if $E_{\lambda-1}$ is empty. So we assume that $E_{\lambda-1} \neq \emptyset$. An edge belongs to $E_{\lambda-1}$ if it connects a vertex $z \in Z$, containing all labels of L, and a vertex $z' \in Z$, containing $|L| - 1$ labels. Notice that no path between z and z' in G can be supporting, except for the edge $\{z, z'\}$ itself. Therefore, edge $\{z, z'\}$ must be in any tree support. □

4 Minimum Tree Supports for Labeled Hypergraphs

In this section we present the Algorithm MINIMUM TREE SUPPORT (MTS) that takes as an input a labeled hypergraph $H(D) = (Z, S(L))$ for the AEDD $D = (L, Z)$, as

Algorithm 1: MINIMUM TREE SUPPORT (MTS)

```
Input: labeled hypergraph $H(D) = (Z, S(L))$ for AEDD $D = (L, Z)$,
edge weight function $w\colon E \to \mathbb{R}$ for $E = \binom{Z}{2}$
Output: minimum tree support $T$ for $H(D)$ or infeasibility notification
1 if $H(D)$ has no tree support then return infeasible
2 partition $E$ into sets $E_i$, $i = 0, \dots, |L| - 1$, of edges with equal cardinality $i$
3 $F \leftarrow \emptyset$
4 for $i \leftarrow |L| - 1$ to $0$ do
5     foreach edge $e = \{u, v\} \in E_i$ in non-decreasing order of weights do
6         if $u$ and $v$ belong to different connected components of $F$ then
7             $F \leftarrow F \cup \{e\}$
8 return $F$
```

well as the weight function $w\colon E \to \mathbb{R}$ for the skeleton $G = (V, E)$ of $H(D)$, and produces a minimum tree support T for $H(D)$. The algorithm grows an initially empty forest $F = \emptyset$ and implements $|L|$ hierarchy steps. Recall that $E_i \subseteq E$ are all edges of G with cardinality i. During step $i = |L| - 1, \dots, 0$, the algorithm adds to F a subset of the edges of E_i, which we denote by F_i. Recall that $E_{\geq i} = \bigcup_{j=i}^{\lambda-1} E_j$. Analogously to this notation, we set $F_{\geq i} = \bigcup_{j=i}^{\lambda-1} F_j$. Thus $F_{\geq i}$ are the edges added to F in the steps $|L| - 1$ down to i. Observe that the check at line 6 ensures that $F_{\geq i}$, $i = |L| - 1, \dots, 0$ is a forest. Recalling the definition of the cardinality of a path, we derive the following:

Property 2. Any two vertices $u, v \in Z$ that are connected by a path of cardinality at least k in G, are connected by a path in $F_{\geq k}$.

In the following we first prove that if $H(D)$ is a tree-hypergraph then the output of the Algorithm MTS is a tree support for $H(D)$ (Lemma 4) and then prove that it is actually a minimum tree support (Lemma 5). We conclude the correctness and analyze the running time of Algorithm MTS in Theorem 1.

Lemma 4. *If $H(D)$ is a tree-hypergraph, then Algorithm MTS computes a tree support of $H(D)$.*

Proof. By induction on $i = \lambda - 1, \dots, 1$, we show that there exists a tree support $T_{\geq i}$ for $H(D)$ that extends the forest $F_{\geq i}$. Observe that the base case follows from Lemma 3. As an induction hypothesis we assume that there exists a tree support $T_{\geq i+1}$ for $H(D)$ that extends the forest $F_{\geq i+1}$. Let $T_{\geq i} \equiv (T_{\geq i+1} \setminus E_{\geq i}) \cup F_{\geq i}$. In order to show that $T_{\geq i}$ is a tree support for $H(D)$ we prove that: (a) $T_{\geq i}$ is a spanning tree of G, (b) $T_{\geq i}$ is a support for $H(D)$.

(a) Consider a connected component C of $T_{\geq i+1} \cap E_{\geq i}$. By Property 2, any two vertices of C are also connected in the forest $F_{\geq i}$. Let C' be a connected component of $F_{\geq i}$, and $e = \{u, v\} \in C'$. By Property 1, u and v are connected in $T_{\geq i+1}$ (which is a tree support) by a path of cardinality at least $c(e)$ and therefore by a path in $T_{\geq i+1} \cap E_{\geq i}$.

Thus, connected components of $T_{\geq i+1} \cap E_{\geq i}$ and $F_{\geq i}$ have the same vertex sets. Therefore, by Lemma 2, we have that $(T_{\geq i+1} \setminus E_{\geq i}) \cup F_{\geq i}$, and therefore $T_{\geq i}$, is a spanning tree of G.

(b) Recall that $T_{\geq i} \equiv (T_{\geq i+1} \setminus E_{\geq i}) \cup F_{\geq i}$. Let $F^C_{\geq i}$ denote a connected component of $F_{\geq i}$ and let G^C denote the subgraph of G induced by the vertices of $F^C_{\geq i}$. We observe that, in order to show that $T_{\geq i}$ is a tree support for $H(D)$, it is enough to show that $F^C_{\geq i}$ is a tree support for the hypergraph induced by G^C. Assume for the sake of contradiction that $F^C_{\geq i}$ is not a tree support for the hypergraph induced by G^C. Then, there exists a minimal non-supporting path $p(u, v, F^C_{\geq i})$. Therefore, there exists a hyperedge s that contains u and v, but does not contain any internal vertex of $p(u, v, F^C_{\geq i})$. Let $S_1, \ldots, S_k$ be a sequence of hyperedge sets defined by path $p(u, v, F^C_{\geq i})$ such that $S_1, \ldots, S_k$ together with a hyperedge set S' containing s define a cycle. Notice that $|S_j| \geq i, \forall j, 1 \leq j \leq k$, since these sets are formed by the path $p(u, v, F^C_{\geq i})$. Also observe that there is no index $j, 1 \leq j \leq k$ such that $S' \subseteq S_j$, since $s \in S'$, and $s \notin S_j$.
Recall from the proof of statement (a), that the connected components of $T_{\geq i+1} \cap E_{\geq i}$ and $F_{\geq i}$ have the same vertex sets. Thus, there exists a path $p(u, v, T_{\geq i+1} \cap E_{\geq i})$. Since $T_{\geq i+1}$ is a tree support, and the sequence $S', S_1, \ldots, S_k$ of hyperedge sets in G forms a cycle, we infer by Lemma 1, that each edge e of $p(u, v, T_{\geq i+1} \cap E_{\geq i})$ supports at least two of these hyperedge sets, one of which is S'. Let S_j, $1 \leq j \leq k$ be the second hyperedge set supported by e. Recall that $S' \not\subseteq S_j$, therefore $|S' \cup S_j| > |S_j|$. Thus, $c(e) \geq |S' \cup S_j| > |S_j| \geq i$, i.e. $c(e) \geq i + 1$, for each $e \in p(u, v, T_{\geq i+1} \cap E_{\geq i})$, implying that $e \in F_{\geq i+1}$. By recalling that $T_{\geq i+1}$ extends $F_{\geq i+1}$ and that $p(u, v, F^C_{\geq i})$ is a path in $F_{\geq i}$, which contains $F_{\geq i+1}$, we conclude that $p(u, v, F^C_{\geq i}) = p(u, v, T_{\geq i+1} \cap E_{\geq i})$. Thus $T_{\geq i+1}$ also contains a non-supporting path $p(u, v, T_{\geq i+1} \cap E_{\geq i})$, which is a contradiction to the induction hypothesis that $T_{\geq i+1}$ is a tree support. □

Lemma 5. *If $H(D)$ is a tree-hypergraph and w an edge-weight function for the skeleton, then the tree support computed by Algorithm MTS is a minimum tree support.*

Proof. The proof is again by induction over the hierarchy steps of Algorithm MTS. We show that after each hierarchy step i there is a minimum tree support that extends the forest $F_{\geq i}$. It is easy to see that this is true after the first hierarchy step $\lambda - 1$ as we know that $F_{\lambda-1} = E_{\lambda-1}$ and that any tree support of $H(D)$ contains $E_{\lambda-1}$ by Lemma 3.

So let $i < \lambda - 1$ and assume by induction that there is a minimum tree support T_{i+1} that extends $F_{\geq i+1}$. Hierarchy step i considers the edge set E_i of edges with cardinality i. If $E_i = \emptyset$ or no edges are added in step i, we have $F_{\geq i} = F_{\geq i+1}$ and the statement holds immediately. So let $E_i \neq \emptyset$ and $F_i \neq \emptyset$. We show that after each edge addition in the current hierarchy step there is a minimum tree support that extends the current forest F assuming that this was true before the edge was added. Let $e = \{u, v\}$ be the next edge to be added by the algorithm and let $\hat{T}$ be a minimum tree support extending the forest F, where $e \notin F$. If $e \in \hat{T}$ there is nothing to show, so assume $e \notin \hat{T}$.

Then $\hat{T} \cup \{e\}$ contains a cycle $\hat{K}$. We further know from Lemma 4 that the final tree T, computed by Algorithm MTS, extends $F \cup \{e\}$ and is a tree support. We show that there is an edge $\hat{e} \in \hat{K} \setminus T$, for which there is a cycle K in $T \cup \{\hat{e}\}$ that contains both e and $\hat{e}$.

Firstly, the set $\hat{K} \setminus T$ is not empty since otherwise T would contain a cycle. Assume that no edge in $\hat{K} \setminus T$ has the desired property. Let $(\{u, v\}, \{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{k-1}, v_k\})$ be the edge sequence of $\hat{K}$, where $v = v_1$ and $u = v_k$, and let $1 \leq f_1 < \cdots < f_l \leq k$ be the indices of all edges $e_{f_j} = \{v_{f_j}, v_{f_j+1}\} \in \hat{K} \setminus T$ $(1 \leq j \leq l)$. If we replace each such edge e_{f_j} by the path $p(v_{f_j}, v_{f_j+1}, T)$ we obtain a (not necessarily simple) cycle in T, which is a contradiction to T being a tree.

So let K be a cycle in $T \cup \{\hat{e}\}$ that contains the edges $e = \{u, v\}$ and $\hat{e} = \{\hat{u}, \hat{v}\}$. Since T is a tree support, all edges of the path $p(\hat{u}, \hat{v}, T) = K \setminus \{\hat{e}\}$ must support the hyperedge set $S(\hat{u} \cap \hat{v})$. In particular, edge e supports $S(\hat{u} \cap \hat{v})$. Analogously, $\hat{T}$ is a tree support and thus all edges of the path $p(u, v, \hat{T}) = \hat{K} \setminus \{e\}$, including the edge $\hat{e}$, support the hyperedge set $S(u \cap v)$. It follows that $u \cap v = \hat{u} \cap \hat{v}$ and thus all edges of $\hat{K}$ support $S(\hat{u} \cap \hat{v})$.

We define the tree $\hat{T}_e = (\hat{T} \setminus \{\hat{e}\}) \cup \{e\}$ that replaces $\hat{e}$ by e and claim that it is also a tree support of $H(D)$. For any two vertices $x, y \in Z$ with $\hat{e} \in p(x, y, \hat{T})$ the hyperedge set that has to be supported by every edge of $p(x, y, \hat{T})$ is $S(x \cap y) \subseteq S(\hat{u} \cap \hat{v})$. Since the edges of path $p(x, y, \hat{T}_e)$ are contained in $p(x, y, \hat{T}) \cup \hat{K}$, they also support $S(x \cap y)$. Thus we have showed that there is a tree support, namely $\hat{T}_e$, that extends $F \cup \{e\}$.

It remains to show that $\hat{T}_e$ is a *minimum* tree support. We first observe that both edges e and $\hat{e}$ have the same cardinality $c(e) = c(\hat{e}) = i$ and thus $e, \hat{e} \in E_i$ are both considered in hierarchy step i of the algorithm. Our algorithm, however, considers e before $\hat{e}$, which means that $w(e) \leq w(\hat{e})$. Since $\hat{T}$ is a minimum tree support by the induction hypothesis and $w(\hat{T}_e) \leq w(\hat{T})$, we obtain that $\hat{T}_e$ is also a minimum tree support. This is true for every edge addition in hierarchy step i, so in particular for $F = F_{\geq i}$ after the last edge addition in this hierarchy step. But this already concludes the inductive argument for the whole hierarchy step and shows together with Lemma 4 that the result $T = F_{\geq 0}$ of algorithm MTS is indeed a minimum tree support of $H(D)$. □

Theorem 1. *Given a labeled hypergraph $H(D)$ with n vertices and m hyperedges for an AEDD D Algorithm MTS computes in $O(n^2 m)$ time a minimum tree support T or reports that no tree support exists.*

Proof. Algorithm MTS starts by checking whether $H(D)$ has a tree support using the feasibility check proposed by Johnson and Pollak [14]. If the test fails the algorithm reports this result; otherwise $H(D)$ is a tree-hypergraph and thus we know by Lemma 5 that the resulting tree T is a minimum tree support. This proves the correctness.

The feasibility test of Johnson and Pollak [14] in line 1 of the algorithm is based on testing whether the dual hypergraph $H(D)^*$ of $H(D)$ is acyclic. The dual hypergraph of $H(D)$ can be constructed in $O(nm)$ time and has a vertex for each hyperedge of $H(D)$ and a hyperedge for each vertex of $H(D)$, which contains all hyperedges incident to that vertex. The acyclicity of $H(D)^*$ can be tested in $O(nm)$ time [20].

The next step in line 2 of the algorithm is to partition the edge set E into subsets based on the edge cardinalities. For each edge $\{u, v\} \in E$ computing the cardinality of the intersection $u \cap v$ takes $O(m)$ time, since each vertex consists of at most m labels. Since we have $O(n^2)$ edges this takes $O(n^2 m)$ time in total.

Finally, in lines 3–8 we run a modified version of Kruskal's algorithm to compute a minimum spanning tree. Unlike the original algorithm, we do not sort the whole edge set E by non-decreasing weights, but rather perform $|L|$ hierarchy steps, in which we consider the edges of each subset E_i in the edge partition separately in non-decreasing weight order. This modification, however, does not affect the running time and thus the last part of Algorithm MTS takes $O(|E| \log |Z|)$ time, just as computing a minimum spanning tree by Kruskal's algorithm. The set E is of size $O(n^2)$ and vertices in Z are subsets of the label set L, i.e., $\log |Z|$ is of size $O(m)$. Thus the total running time of Algorithm MTS is $O(n^2 m)$. □

5 Minimum Tree Supports for Hypergraphs

Labeled hypergraphs, in particular for abstract Euler diagram descriptions as considered in the previous section, seem to be of limited interest at first sight. So it is a natural question to ask for a minimum tree support of a general tree-hypergraph $H = (V, S)$. As discussed in Section 1, Korach and Stern [16] showed that this problem can be solved efficiently in $O(n^4 m^2)$ time, where $n = |V|$ and $m = |S|$.

In this section we generalize Theorem 1 to arbitrary hypergraphs, and thus improve the best known running time from $O(n^4 m^2)$ to $O(n^2(m + \log n))$. The tool to achieve this is to define a mapping that transforms an arbitrary hypergraph to an equivalent labeled hypergraph so that we can apply Algorithm MTS.

Theorem 2. *Given a hypergraph H with n vertices and m hyperedges and an edge weight function $w\colon \binom{V}{2} \to \mathbb{R}$ we can compute in $O(n^2(m + \log n))$ time a minimum tree support T of H or report that no tree support exists.*

Proof. An important difference between an arbitrary hypergraph H and the labeled hypergraph $H(D)$ for an AEDD D is that $H(D)$ contains at most one zone for each possible subset of labels, whereas H may contain any number of vertices that have exactly the same hyperedge incidences. This forces us to slightly modify Algorithm MTS and its analysis.

Let $H = (V, S)$ be a hypergraph. We start by describing the mapping μ, which maps H to an equivalent labeled hypergraph. We define the label set $L_S = \{l_1, \dots, l_m\}$, which contains one unique label $l_i = l(s_i)$ for each hyperedge $s_i \in S$. Each vertex $v \in V$ is mapped to an indexed label set $(v, \mu(v)) = (v, \{l(s_i) \mid v \in s_i\})$, where $\mu(v)$ contains the labels of all hyperedges containing v. We explicitly allow that two distinct vertices $v \neq v'$ are mapped to the same label set $\mu(v) = \mu(v')$, but their indexed label sets $(v, \mu(v))$ and $(v', \mu(v'))$ are distinguishable. Similarly, we map each hyperedge $s \in S$ to a set of indexed label sets $\mu(s) = \{(v, \mu(v)) \mid v \in s\}$. We use the notation $\mu(V)$ to denote the set $\{(v, \mu(v)) \mid v \in V\}$ and $\mu(S)$ to denote the set $\{\mu(s) \mid s \in S\}$. This defines a labeled hypergraph $\mu(H) = (\mu(V), \mu(S))$, which is isomorphic to the labeled hypergraph of the AEDD $D = (L_S, \mu(V))$ if no two vertices in V are incident to exactly the same hyperedges. We further define a new edge weight function $\mu(w)$ as $\mu(w)((u, \mu(u)), (v, \mu(v))) = w(u, v)$.

Since our construction simply replaces each vertex of H by an indexed label set indicating its incident hyperedges it is obvious that each tree support of H is in one-to-one correspondence to a tree support of $\mu(H)$, in particular an MTS of $\mu(H)$ is also an

MTS of H. Thus we can apply Algorithm MTS to the labeled hypergraph $\mu(H)$ and obtain a minimum tree support T for H.

We need to pay attention to one minor issue in the correctness proof of the algorithm. In Section 4 the base case of the inductive proofs started with edges of cardinality $m-1$. Now we might have edges of cardinality m, namely if multiple vertices are contained in every hyperedge in S. If we run Algorithm MTS with an extra hierarchy step for the cardinality-m edges it computes a minimum spanning tree of the vertex set $\mu(V_m) = \{(v, \mu(v)) \in \mu(V) \mid \mu(v) = S\}$. Using the fact shown by Korach and Stern [16] that every element of the hyperedge intersection closure of H (which contains S and all intersections of subsets of S) induces a connected subtree in every tree support of H, we know that every minimum tree support of $\mu(H)$ must contain a minimum spanning tree of $\mu(V_m)$. This serves as the new base case of the induction; the remainder of the correctness proofs in Section 4 continues to hold.

For the running time analysis we again need to pay attention to a small detail related to vertices with the same label set. Lines 3–8 of Algorithm MTS are a modification of Kruskal's algorithm and thus need $O(n^2 \log n)$ time on a complete graph with n vertices. But since we may have more than one vertex with the same label set it is no longer true in general that $\log n \in O(m)$. Thus the modification of Algorithm MTS takes $O(n^2(m + \log n))$ time to compute the MTS T for $\mu(H)$.

Finally, it remains to argue that $\mu(H)$ can be computed in the same time bound. For creating $\mu(V)$ and $\mu(S)$ we simply scan all hyperedges in S and append their labels to the contained vertices. This can be done in $O(nm)$ time since each hyperedge contains $O(n)$ vertices. Hyperedges in $\mu(S)$ are not explicitly represented as sets of label sets, but rather as sets of pointers to the vertices in $\mu(V)$. □

6 Conclusion

We have studied the problem of computing minimum tree supports for hypergraphs and we have seen that our algorithm for the special case of labeled hypergraphs induced by abstract Euler diagram descriptions easily generalizes to arbitrary hypergraphs. We improved the previously best known running time for computing minimum tree supports [16] from $O(n^4m^2)$ to $O(n^2(m + \log n))$. Moreover, we described an application of minimum tree supports for generating area-proportional Euler diagrams with convex zones and minimum internal concurrencies for abstract Euler diagram descriptions with a labeled tree-hypergraph.

Other types of sparse supports like outerplanar supports give rise to interesting open questions. For example, the complexity of deciding whether a given hypergraph has an outerplanar support is open [5]. On the practical side, it is interesting to study algorithms for generating well-formed Euler diagrams based on outerplanar supports or other larger classes of supports.

References

1. D. Angluin, J. Aspnes, and L. Reyzin. Inferring social networks from outbreaks. In *Algorithmic Learning Theory*, vol. 6331 of *LNCS*, pp. 104–118. Springer, Heidelberg 2010.

2. D. Angluin, J. Aspnes, and L. Reyzin. Network construction with subgraph connectivity constraints. *J. Comb. Optim.*, 2013.
3. U. Brandes, S. Cornelsen, B. Pampel, and A. Sallaberry. Blocks of hypergraphs applied to hypergraphs and outerplanarity. In C. S. Iliopoulos and W. F. Smyth, editors, *Combinatorial Algorithms (IWOCA'10)*, vol. 6460 of *LNCS*, pp. 201–211. Springer, Heidelberg, 2011.
4. K. Buchin, M. van Kreveld, H. Meijer, B. Speckmann, and K. Verbeek. On planar supports for hypergraphs. Technical Report UU-CS-2009-035, Utrecht University, 2009.
5. K. Buchin, M. van Kreveld, H. Meijer, B. Speckmann, and K. Verbeek. On planar supports for hypergraphs. In D. Eppstein and E. R. Gansner, editors, *Graph Drawing (GD'09)*, vol. 5849 of *LNCS*, pp. 345–356. Springer, Heidelberg, 2010.
6. J. Chen, C. Komusiewicz, R. Niedermeier, M. Sorge, and O. Suchý. Effective and efficient data reduction for the subset interconnection design problem. In *Algorithms and Computation (ISAAC'13)*, vol. 8283 of *LNCS*, pp. 361–371. Springer, Heidelberg, 2013.
7. G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg. Constructing scalable overlays for pub-sub with many topics. In *Principles of distributed computing (PODC'07)*, pp. 109–118, 2007.
8. S. Chow. *Generating and Drawing Area-Proportional Euler and Venn Diagrams*. PhD thesis, University of Victoria, 2007.
9. S. Chow and F. Ruskey. Drawing area-proportional Venn and Euler diagrams. In G. Liotta, editor, *Graph Drawing (GD'03)*, vol. 2912 of *LNCS*, pp. 466–477. Springer, Heidelberg, 2004.
10. D.-Z. Du and D. F. Kelley. On complexity of subset interconnection designs. *J. Global Optim.*, 6:193–205, 1995.
11. H. Fan, C. Hundt, Y.-L. Wu, and J. Ernst. Algorithms and implementation for interconnection graph problem. In *Combinatorial Optimization and Applications (COCOA'08)*, vol. 5165 of *LNCS*, pp. 201–210. Springer, Heidelberg, 2008.
12. J. Flower, A. Fish, and J. Howse. Euler diagram generation. *J. Visual Languages and Computing*, 19(6):675–694, 2008.
13. J. Hosoda, J. Hromkovič, T. Izumi, H. Ono, M. Steinová, and K. Wada. On the approximability and hardness of minimum topic connected overlay and its special instances. *Theoretical Computer Science*, 429:144–154, 2012.
14. D. S. Johnson and H. O. Pollak. Hypergraph planarity and the complexity of drawing Venn diagrams. *J. Graph Theory*, 11(3):309–325, 1987.
15. M. Kaufmann, M. van Kreveld, and B. Speckmann. Subdivision drawings of hypergraphs. In I. G. Tollis and M. Patrignani, editors, *Graph Drawing (GD'08)*, vol. 5417 of *LNCS*, pp. 396–407. Springer, Heidelberg, 2009.
16. E. Korach and M. Stern. The clustering matroid and the optimal clustering tree. *Mathematical Programming*, 98(1-3):385–414, 2003.
17. E. Korach and M. Stern. The complete optimal stars-clustering-tree problem. *Discrete Applied Mathematics*, 156:444–450, 2008.
18. P. Rodgers, L. Zhang, and A. Fish. General Euler diagram generation. In G. Stapleton, J. Howse, and J. Lee, editors, *Theory and Application of Diagrams (DIAGRAMS'08)*, vol. 5223 of *LNCS*, pp. 13–27. Springer, Heidelberg, 2008.
19. G. Stapleton, P. Rodgers, and J. Howse. A general method for drawing area-proportional Euler diagrams. *J. Visual Languages and Computing*, 22(6):426–442, 2011.
20. 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 J. Comput.*, 13(3):566–579, 1984.