A Data Structure for Maintaining Acyclicity in Hypergraphs

Percy Liang
MIT CSAIL
Cambridge, MA, USA
pliang@mit.edu

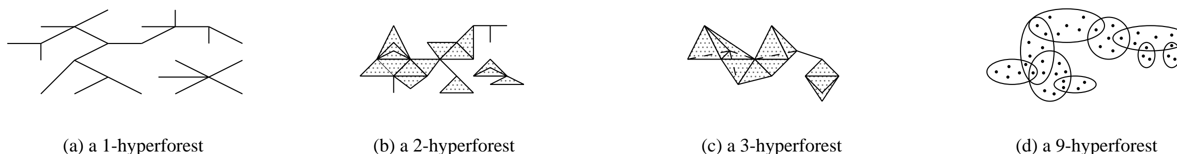
Nathan Srebro
Univ. of Toronto Dept. of CS
Toronto, ON, Canada
nati@cs.toronto.edu

Abstract

Acyclicity is an important property of hypergraphs which has applications in many areas such as graphical models and relational databases. Our contributions in this paper are two-fold: First, we present two new characterizations of a *hyperforest* (equivalently, acyclic hypergraph or triangulated graph) through a hierarchical decomposition of the hyperforest and through the lack of *hypercycles*, a concept defined in this paper. Second, we present the first efficient dynamic data structure for maintaining acyclicity in a hypergraph. The data structure uses as a building block Tarjan's Union-Find data structure (which can be used to maintain acyclicity in graphs) to achieve an amortized expected query time that has an inverse Ackermann dependence on the number of vertices. To demonstrate the practicality of this data structure, we conduct experiments using our data structure to construct high-weight hyperforests.

1 Introduction

Acyclic hypergraphs, or *hyperforests* (Figure 1 shows several examples), are a natural generalization of forests. They have been independently and equivalently defined in many different domains and are also studied as triangulated graphs (hyperforests are hypergraphs formed by the cliques of triangulated graphs). Hyperforests are useful in many domains where higher-order relations are to be captured but certain tree-like "acyclic" properties are also desired.



(a) a 1-hyperforest (b) a 2-hyperforest (c) a 3-hyperforest (d) a 9-hyperforest

Figure 1: Examples of hyperforests of various tree-widths.

The acyclicity provided by hyperforests allows many calculations to be carried out efficiently using dynamic programming. Such calculations include a broad class of combinatorial problems [Cou90] as well as inference in graphical models [Bes74]. In these applications, while the computation is often exponential in the *tree-width* of the hyperforest (which corresponds to the maximum size of the hyperedges, or cliques in a triangulated graph), it is polynomial (often only linear) in the number of vertices the hyperforest. The class of K-hyperforests (having tree-width at most K) is then of particular interest.

When choosing to use K-hyperforests for the above reasons, it is often desirable to find the *best* K-hyperforest, where the quality of a hyperforest is equal to the sum of precomputed weights on its hyperedges. This criterion leads to the problem of finding a maximum-weight K-hyperforest [KS01]. The special case where $K = 1$ is equivalent to the problem of finding a minimum spanning tree (MST). This problem appears as constructing maximum likelihood Markov trees (Chow-Liu trees) [CL68], Hunter-Worsley trees for Bonferroni inequalities [Wor82], and trees used to ensure efficient combinatorial op-

timization (e.g. [Mat99]). Generalizations to higher tree-width hyperforests are possible, desirable, and have recently been investigated [Mal91, Sre01, BP01, Tom86].

Unfortunately, when $K > 1$, finding the maximum weight K-hyperforest is NP-complete and finding good approximation algorithms remains an open problem [Sre00]. A common heuristic is a Prim-like greedy approach, in which hyperedges containing a single new vertex are incrementally added to a fully-connected hypertree [Mal91, BP01, BJ02]. Alternatively, one might consider a more flexible and possibly more powerful Kruskal-like greedy approach, in which hyperedges are added to a possibly unconnected K-hyperforest. In order to do so, it is necessary to ensure that a candidate hyperedge would not break the acyclicity of the existing hyperforest. A particular situation where the Kruskal-like approach is necessary is when we would like to greedily augment an initial, possibly unconnected, hyperforest which is required or strongly desired in our final hyperforest. For example, we may want to augment a high-weight substructure found by global search techniques (it is possible to efficiently find hyperforests containing at least a constant fraction of the optimal weight [KS01]).

Also in other situations, one might wish to add hyperedges to a hyperforest, ensuring that acyclicity is maintained. For example, acyclicity is important in precluding possible conflicts in relational databases [BFMY83] and one might want to ensure that adding new relations to an existing database scheme maintains acyclicity.

When augmenting forests, Tarjan's Union-Find dynamic data structure can be used to efficiently check if a candidate edge would break the acyclicity by keeping track of the connected components in the graph. The main result in this paper is a dynamic data structure for hyperforests that serves a purpose analogous to Tarjan's Union-Find structure for forests: for any candidate hyperedge, the data structure can verify that adding the hyperedge will not break the acyclicity of the hyperforest.

Another product of this paper is the study acyclicity in hypergraphs. We show that in hypergraphs, it is no longer enough to consider incidence or reachability between two hyperedges as binary properties as it is in graphs. Instead, we present a novel hierarchical view of hyperforests, where each level captures a different extent of reachability. Although acyclic hypergraphs have been studied for three decades using many equivalent characterizations, we are not aware of any definitions of a *hypercycle*, a certificate that a hypergraph is not acyclic; this paper provides such a concept. These two characterizations provide deeper insight into the nature of acyclicity and help us in constructing and proving correctness of the data structure we present.

The rest of this paper is organized as follows: in Section 2 we define hyperforests and specify the desired data structure. In Section 3 we provide two new characterizations of acyclicity based on a hierarchical decomposition of a hyperforest and on the absence of hypercycles. After laying the foundations of acyclicity, we present the data structure in Section 4. Finally, in Section 5 we give experimental results on using the data structure in a Kruskal-like approach to find high-weight hypertrees, evaluating both the efficiency of the algorithm and quality of its solutions.

2 Hypergraphs and acyclicity

Preliminaries A *hypergraph* H over a set of vertices V is a collection of *hyperedges*, where each hyperedge is a subset of V. In particular, a k-hyperedge is a set of $k+1$ vertices, and a k-hypergraph is one in which each of its hyperedges contains at most k+1 vertices. We call k the *order* of the hypergraph. Note that standard (undirected) graphs are 1-hypergraphs, where each edge is a 1-hyperedge containing 2 vertices.

A hyperedge h_1 can be a superset of another hyperedge h_2; in this case, we say that h_1 *covers* h_2. A *maximal* hyperedge in a hypergraph is one that is only covered by itself. In this paper, we consider two hypergraphs to be equivalent if they have the same set of maximal hyperedges (or equivalently, same set of covered hyperedges). One can think of the hypergraph implicitly containing all its covered hyperedges. However, in this paper, when we refer to the hyperedges of a hypergraph, we are talking

about only the maximal hyperedges. Covering also applies to entire hypergraphs: a hypergraph H_1 *covers* a hypergraph H_2 if each hyperedge in H_2 is covered by some hyperedge in H_1.

We now define the property of a hypergraph that is central to this paper: *(hyper)acyclicity*. There are several equivalent definitions of hypergraph acyclicity in common use (see [Sre00] for a review). Here, we define it using the classic notion of a *tree structure* (essentially the same as tree decomposition, junction tree, join tree, etc.). In Section 3, we introduce two new definitions of acyclicity which will lead up to the analysis of our data structure.

Definition 1 (Tree structure of a hyperforest). *A hypergraph H has a* **tree structure** $T(H)$ *if and only if T is a tree whose vertices are the hyperedges in H, and the following* **running intersection property** *holds: if $h_1, h_2, \ldots, h_m$ is a path of hyperedges in T, then $h_1 \cap h_m$ is covered by every hyperedge h_i on the path. A hypergraph with a tree structure is said to be* **acyclic** *and is referred to as a* **hyperforest**.

For example, the tree structure of a standard tree T ($k = 1$) is a tree whose nodes are the edges of T and edges are nodes of T with degree at least 2.

Definition 2 (Tree-width). *The* **tree-width** *of a hyperforest H is the order of H (the size of the largest hyperedge in H minus 1). The tree-width of a general hypergraph H is the minimum tree-width of a hyperforest that covers H.*

As mentioned in the introduction, tree-width and (hyper)acyclicity can be equivalently studied by considering cliques in standard graphs. For any graph, consider its clique-hypergraph: a hypergraph whose hyperedges are the cliques of the original graph. A hypergraph is acyclic if and only if it is a clique-hypergraph of a triangulated graph. The tree-width of a hypergraph is the same as the tree-width of the corresponding graph, which is one less than the the maximum clique size of an optimal triangulation of the graph.

Data structure specifications This paper presents the first dynamic data structure that allows one to augment an existing hyperforest by adding hyperedges to it, while checking that it remains acyclic. Specifically, the data structure should keep track of the "current" hyperforest H and support the following two operations:

QUERY(h_{new}): returns TRUE iff $H \cup \{h_{\text{new}}\}$ is acyclic.

INSERT(h_{new}): augments $H \leftarrow H \cup \{h_{\text{new}}\}$, assuming that the resulting H is acyclic.

A technical detail is that the data structure should allow for adding hyperedges that cover previously added hyperedges. Any covered hyperedges should simply be subsumed by the new hyperedge.

Alternative goal: low tree-width Unlike forests, hyperforests do not form a monotone family of hypergraphs: removing hyperedges from an acyclic hypergraph might make it cyclic, and conversely, adding hyperedges to a cyclic hypergraph might make it acyclic (see Figure 2).

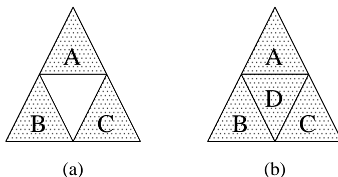


Figure 2: An example illustrating the non-monotonicity of acyclicity in hypergraphs. (a) is not acyclic, but adding the 2-hyperedge D to (a) results in (b), which is acyclic.

If we would like to do efficient computations on a cyclic hypergraph, it is often admissible to add extra hyperedges to a hypergraph to obtain a covering hyperforest (equivalent to triangulating the corresponding graph). We can then perform dynamic programming over the tree structure of that hyperforest.

In such a situation, we might want data structure that checks whether adding a hyperedge maintains a hypergraph of tree-width at most k rather than our data structure, which strictly maintains acyclicity. Our data structure imposes stronger constraints: if all hyperedges added to our data structure are of order at most k, then the tree-width of the resulting hyperforest will be at most k. However, the converse is not true: our data structure might reject a hyperedge h_{new} even though adding it maintains tree-width k. For example, in Figure 2, if our current hyperforest was $\{A, B\}$, our data structure would reject C, even though the tree-width of $\{A, B, C\}$ is 2. We would have to add D first before adding C.

Before considering dynamic data structures for maintaining low tree-width, it is important to remember that even computing the tree-width of a graph statically, or equivalently finding a narrow triangulation, is by itself a very difficult task. Although linear time algorithms for constant tree-width have been proposed [Bod96], the dependence on the tree-width is extremely prohibitive and these algorithms are not usable in practice. Instead, various heuristics, approximation algorithms, and super-polynomial-time algorithms are used [SG97].

3 A new look at hyperforests

Thus far, we have defined acyclicity in terms of the existence of a tree structure over the hyperedges. There are two missing aspects of this definition that we will remedy in this section. First, a given hyperforest may have many different tree structures, which makes talking about a concrete structure difficult. Section 3.1 addresses this issue by defining a single hierarchical structure for any hyperforest. Second, cyclicity is currently defined through the absence of a tree structure. Section 3.2 defines the new concept of a hypercycle, which generalizes cycles in graphs and provides a certificate of cyclicity in hypergraph. These two equivalent definitions of acyclicity provide additional insight into the precise nature of hyperforests. In Section 4, these concepts will be instrumental in the analysis of our data structure.

3.1 Hierachical structure of a hyperforests

The interaction between two hyperedges in a hypergraph is substantially more complex than the interaction between two edges in a standard graph. This complication is what makes hyperacyclicity a more elusive concept to analyze. In a graph, two edges are either incident or disjoint. In a k-hypergraph, there are $k+1$ extents to which two hyperedges can overlap, corresponding to overlap sizes ranging from 0 to k. In this section, a hierarchical decomposition of a hypergraph into *superedges* will allows us to concentrate on one extent of overlap at a time. We begin with the basic definition of a superedge, which attempts to clump together hyperedges:

Definition 3 (Superedge). *Two hyperedges h_1 and h_m are k-reachable if there exists a sequence of hyperedges $h_1, h_2, \ldots, h_m$ for which each overlap $h_i \cap h_{i+1}$ is at least size k. A* $k-1$**-superedge** *q of a hyperforest H is a maximal k-reachable subset of H (every pair of hyperedges in q are k-reachable from each other, but no other hyperedges are k-reachable).*

For convenience, denote the set of vertices in the union of the hyperedges in a superedge q as $\tilde{q} = \cup q$.

The superedges specify a hierarchical partitioning of the hyperedges of a hypergraph: a k-superedge consists of a set of $k+1$-superedges. The root of the hierarchy is the single -1-superedge, which contains all the hyperedges. The next level in the hierarchy are the the 0-superedges, which correspond to the connected components. At the bottom of the hierarchy, each K-superedge contains exactly one hyperedge, where K is the maximium order of a hyperedge. It is important to note that any hypergraph, not only hyperforests, can be decomposed into a hierarchy of superedges.

With this hierarchical partitioning of the hyperedges, we can now look at a K-hypergraph one *level* at a time (Figure 3 shows an example 3-hyperforest). At level k, we focus on a single $k-1$-superedge

p and the k-superedges inside p. Within p, $< k$-reachability does not exist because all hyperedges are k-reachable (at least), and $> k$-reachability has been abstracted into the individual k-superedges, so we mainly concern ourselves with k-reachability. We can think of the k-superedges as fancier k-hyperedges. In a way, a k-superedge q "functions" as a k-hyperedge because all overlaps between q and other k-superedges have size at most k. Even though we have tried to focus on only k-reachability, the overlap between two superedges can still be complicated and requires some attention:

Definition 4 (Simple overlap of superedges). *The* overlap *between two superedges q_1 and q_2 of H is $s = \tilde{q}_1 \cap \tilde{q}_2$. q_1 and q_2 are said to* overlap simply *if s is the intersection of some $h_1 \in q_1$ and $h_2 \in q_2$.*

Acyclicity and incidence graph structures We now have the necessary tools to properly define the unique structure corresponding to a hypergraph that we will analyze to determine acyclicity:

Definition 5 (Incidence graph structure). *The* **incidence graph structure** *of a $k-1$-superedge containing a set of k-superedges is a bipartite graph whose left vertex set consists of the k-superedges and the right vertex set consists of all maximal overlaps between any two superedges. An edge connects a k-superedge q with an overlap s if $s \subset \tilde{q}$.*

A hypergraph has an incidence graph structure for each superedge of the hypergraph. Figure 3 shows the incidence graph structures for an example hypergraph.

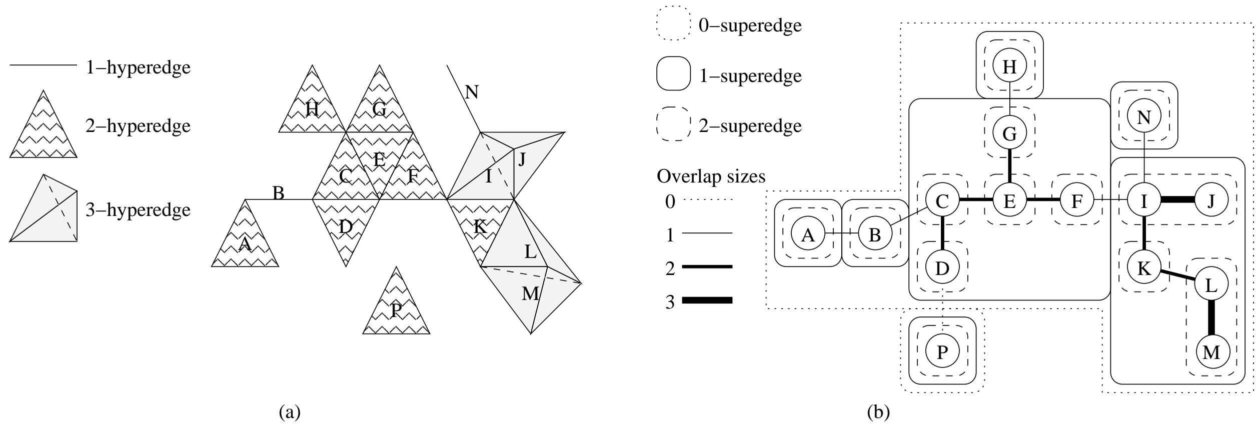


Figure 3: An example of how the 3-hypergraph in (a) (which happens to be acyclic) can be viewed in terms of its superedges and incidence graph structure (b).

We next show what acyclicity means in terms of these structures. But first, a lemma will reveal the relation between a tree structure of a hyperforest and the superedges of the hyperforest.

Lemma 6 (Superedges and tree structures). *The hyperedges in any superedge of a hyperforest H correspond to a contiguous subtree of nodes in any tree structure of H.*

Proof. Let T be any tree structure of a hyperforest H, and q be any $k-1$-superedge for any k. Consider the path in T between any two hyperedges $h_1, h_2 \in q$. The overlap s between any two successive hyperedges on this path separates h_1 and h_2, and since any two hyperedges in q are k-reachable, s must have size at least k. Therefore, the path must contain only hyperedges from q. Put it another way, each superedge occupies a contiguous portion of the nodes in T. □

Definition 7 (Simply acyclic incidence graph structures). *The incidence graph structure of a $k-1$-superedge p is said to be* **simply acyclic** *if (1) each pair of k-superedges in p overlap simply, (2) the incidence graph is acylic, and (3) the running intersection property holds, i.e. the overlap between the first and last k-superedges of any path of k-superedges through the incidence graph is contained in every k-superedge on the path.*

This definition is similar to that of tree structures, but has the important distinction of being a property of the unique incidence graph structure, which always exists. Contrast this with tree structures, where for a given hypergraph, there could be 0, 1, or many.

Theorem 8. *A hypergraph is acyclic if and only if the incidence graph structure of every superedge is simply acylic.*

Proof.

$\Longrightarrow$: We will show that if a hypergraph H is acyclic, and thus has a tree structure T, then the incidence graph structure of each of its superedges is simply acyclic. Let p be a $k-1$-superedge.

1. By Lemma 6, for any pair of k-superedges q_1, q_2 in p, there exists a unique pair of hyperedges $h_1 \in q_1$ and $h_2 \in q_2$ which are closest to each other in T. Furthermore, every path between a hyperedge in q_1 and a hyperedge in q_2 must go through both h_1 and h_2. By the running intersection property of Definition 1, it follows that $h_1 \cap h_2 = \tilde{q}_1 \cap \tilde{q}_2$. Thus, all overlaps between two k-superedges are simple.
2. We now show that the incidence graph structure is acyclic. Since superedges occupy contiguous portions of T, clumping together hyperedges belonging to the same superedge results in a tree structure T_p over k-superedges (T is a minor of the graph T_p). Note that the edges of T_p correspond to the overlaps (which we have shown to be simple) between k-superedges. The running intersection property guarantees that all maximal overlaps between superedges are represented by one of these edges, and that edges corresponding to the same overlap will exist in a contiguous subtree of T_p. Thus, we can convert T_p into the unique incidence graph structure of p as follows (Figure 4(b) to (c)): For each maximal overlap s, let Q_s be the set of k-superedge nodes that contain s. The set Q_s contains exactly the k-superedge nodes incident on some edge corresponding to s. Connect each k-superedge node in Q_s to a new overlap node corresponding to s, and disconnect the k-superedge nodes from each other. By construction, this graph is acyclic.
3. The running intersection property holds due to the same property holding in T.

(a) hyperforest (b) tree structure (c) graph incidence structure

Figure 4: An example of the tree structure (b) and graph incidence structure (c) of a hyperforest (a).

$\Longleftarrow$: We will inductively construct a tree structure T from a simply acylic incidence graph structure. Having constructed a tree structure for each k-superedge, let us now construct a tree structure for the k–1-superedges, essentially reversing the process from the first half of the proof. Let p be a $k-1$-superedge. For each overlap s in the incidence graph structure, let $q_1, \ldots, q_m$ be the superedges adjacent to s. Since all overlaps are simple, we can find hyperedges $h_1, \ldots, h_m$ from the corresponding superedges that each contain s. Arbitrarily join these hyperedges with any spanning tree. If this process is done for each overlap s, the resulting structure will indeed by acyclic. The running intersection property follows from the same property holding for the incidence graph structure on p. □

3.2 Hypercycles

While a tree structure is a certificate that proves the acyclicity of a hypergraph, we know of no previous work describing a certificate for cyclicity. In this section, we introduce such a certificate, a *hypercycle*, which generalizes naturally from a cycle of a graph.

To appreciate the intricacies involved of such a definition, here is a reasonable but failed attempt to define a hypercycle in terms of hyperedges: a hypercycle is a sequence of maximal hyperedges $h_1, \ldots, h_m$ with distinct overlaps between successive hyperedges (including $h_1 \cap h_m$). Maximality is required to exclude hypercycles which are covered by some other hyperedge (thus eliminating cyclicity). Distinctness is required to exclude classifying the edges in a star graph as a hypercycle. But still, A-B-C in Figure 2(b) would be classified as a hypercycle under this definition, but clearly, that hypergraph is acyclic. The problem lies in the fact that larger overlaps sometimes invalidate a view consisting of smaller overlaps which may suggest cyclicity. The solution we propose is based on superedges rather than hyperedges.

Definition 9 (Hypercycle). *A k-hypercycle is one of the following:*

1. k-hyperdoublet*: Two k-superedges that overlap non-simply.*
2. Regular k-hypercycle*: A sequence of at least 3 distinct k-superedges $q_1, q_2, \ldots, q_m$ with distinct overlaps $s_1, \ldots, s_m$ of size exactly k, where $s_i = \tilde{q}_i \cap \tilde{q}_{i+1}$ for $1 \leq i < m$ and $s_m = \tilde{q}_1 \cap \tilde{q}_m$.*
3. Irregular k-hypercycle*: A sequence of k-superedges as defined above, except that $|s_m| < k$ but s_m is not a subset of any of the other overlaps.*

Hypercycles as we have defined them only correspond to the analogue of simple cycles: there are no overlap repetitions allowed. A regular 1-hypercycle is exactly a simple cycle: overlaps between edges of the cycle are the distinct vertices along it, while the overlap s_m is the vertex between the first and last edges (first example in Figure 5a). Regular hypercycles are straightforward generalizations of cycles. But two other types of k-hypercycles manifest themselves with larger k due to the complex interaction of hyperedges, even when we have tried to partition hyperedges nicely into superedges. A hyperdoublet is analogous to two nodes connected by two distinct edges in a multigraph. An irregular hypercycle can be thought of as a self-loop.

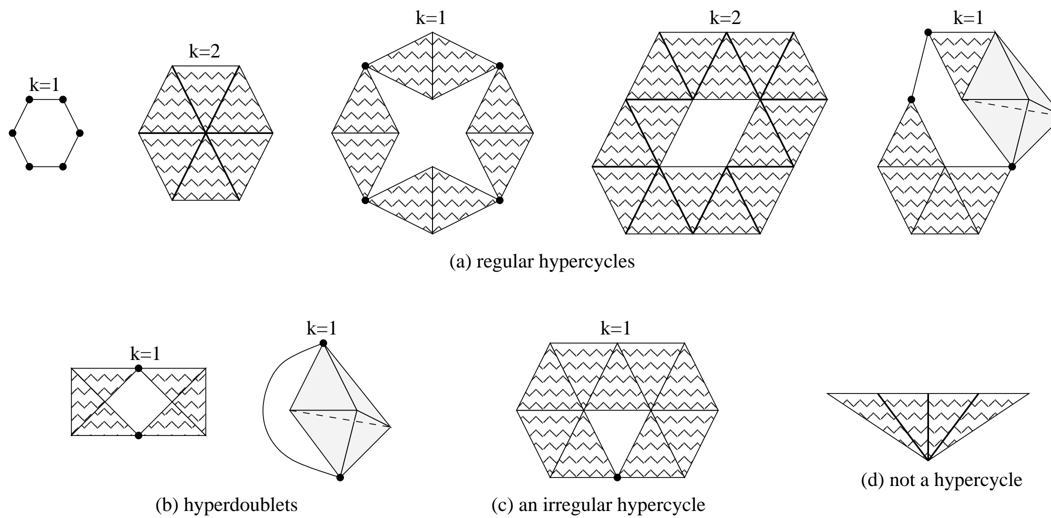


Figure 5: Examples of hypercycles.

Theorem 10. *A hypergraph H is acyclic if and only if H contains no hypercycles.*

Proof. First, note that if a k-hypercycle exists, then one must exist in a $k-1$-superedge. Hyperdoublets correspond to non-simply overlapping superedges. The overlaps in a regular hypercycle are of size k and hence are maximal, so a regular hypercycle corresponds to a cycle in the incidence graph structure of the $k-1$-superedge. An irregular hypercycle either corresponds to a cycle if the overlap s_m between the first and last superedges is maximal. If s_m is not maximal, the running intersection property is

violated since s_m is not contained in all superedges on the path $q_1, s_1, \ldots, s_{m-1}, q_m$ in the incidence graph structure. □

4 The data structure

Having established a powerful characterization of acyclicity in hypergraphs, we now present the main result of this paper, the dynamic data structure that checks acyclicity.

4.1 Union-Find applied to cycle detection

Tarjan's Union-Find data structure is used to keep track of membership in disjoint sets. It supports two operations: querying the set of a member and uniting two sets. The amortized run-time for m operations in $O(\alpha(m, n))$ where $\alpha(\cdot)$ is the sub-logarithmic inverse Ackermann function and n the number of elements.

The data structure can be used to keep track of connected components: querying two vertices can establish if they are in the same connected component or not, and adding an edge between connected components unites them. These operations can be used to maintain acyclicity in standard graphs by ensuring that the endpoints of a candidate edge are in different connected components, satisfying the specifi cations in Section 2.

The Union-Find data structure maintains the simple notion reachability between nodes via edges. But for K-hypergraphs, this is not enough; recall from Section 3.1 that there are K extents of overlap between hyperedges, resulting in K extents of reachability. Superedges allow us to focus on one level at a time. Putting these ideas together, we take the following approach: use the classic Union-Find structure as a building block at each of the K levels. The Union-Find structure at level k shall keep track of k-reachability information between k-overlaps via k-superedges.

4.2 Overview of operations and state

Notation We continue using the term *k-hyperedge* to mean a maximal hyperedge of size $k+1$and *k-overlap* to mean the intersection of two (maximal) hyperedges, which has size k. Each overlap can also be thought of as a covered non-maximal hyperedge that is associated with the two hyperedges whose intersection is the overlap. In addition, we will now use the term *k-supervertex* to denote a covered hyperedge containing k vertices. The intention is to draw an analogy between superedges connecting supervertices and edges connecting vertices. Hyperedges, overlaps, and supervertices are all sets of vertices—they differ only in their role in our proofs.

Data structure state We now explain how our data structure meets the specifi cations of Section 2. The state representing the current K-hyperforest H consists of two parts: hash tables $S_1, \ldots, S_K$ mapping supervertices to identifi ers and K Union-Find structures $U_1, \ldots, U_K$. Each U_k stores disjoint sets of k-supervertex identifi ers, where two k-supervertices belong to the same set if they are k-reachable.

Operations (See Figure 6 for pseudocode) Recall that QUERY(h_{new}) should return TRUE if and only if $H \cup \{h_{\text{new}}\}$ is acyclic . This is done by returning FALSE when there exists some k-supervertices s and t which are connected "outside" of h_{new}, so that adding h_{new} would close a hypercycle. The "outside" criteria is important, since if s and t are only k-reachable "inside" h_{new}, h_{new} would coincide with the corresponding path between s and t and fail to form a hypercycle.

To be able to test k-reachability inside h_{new}, upon each invocation of QUERY, we build K additional Union-Find structures $Z_1, \ldots, Z_K$, local to the QUERY operation (they are created when performing the operation and deleted when the operation is completed). The Union-Find structure Z_k

```
QUERY(h_new):

1  Z_1, ..., Z_K ← ∅, ..., ∅
2  for k ← 1 to K do
3    for s : k-supervertices of h_new do
4      if s ∈ S_k then
5        INSERT(s) using {Z_k} instead of {U_k}
6  for k ← 1 to K do
7    for s, t : k-supervertices of h_new do
8      if U_k(s, t) and not Z_k(s, t) then
9        return FALSE
10 return TRUE

Time: O(4^K (K + α(m, n)))
```

```
INSERT(h_new):

1  assert QUERY(h_new)
2  for k ← 1 to K do
3    s ← an arbitrary k-supervertex of h_new
4    for t : k-supervertex of h_new do
5      union (s, t) in U_k
6      S_k[t] ← |S_k|

Time: O(2^K (K + α(m, n)))
```

```
Data structure state:
   Union-Find structures U_1, ..., U_K
   Hash tables S_1, ..., S_K
Space: O(2^K n)
```

Figure 6: Pseudocode for QUERY and INSERT. The runtimes are for m operations on a K-hyperforest with n vertices. QUERY: first compute $\{Z_k\}$, which store reachability information for $H(h_{\text{new}})$, the projection of H onto h_{new}; for each pair of k-supervertices in h_{new}, test whether they are k-reachable "outside" h_{new}. INSERT: unite all k-supervertices of h_{new} (for each k) and add them to the hash tables.

stores k-reachability information of k-supervertices in the projection $H(h_{\text{new}})$ of H onto h_{new} (those k-supervertices of H which are covered by h_{new}). It is analogous to U_k, which stores k-reachability information of k-supervertices in the entire hypergraph H.

If QUERY(h_{new}) returns TRUE, we may call INSERT(h_{new}) to update the data structure state to reflect $H \cup \{h_{\text{new}}\}$. For each k, we update U_k by uniting all k-supervertices of h_{new} and add all k-supervertices to the hash table S_k.

Complexity Both Union-Find structures and hash sets require space linear in the number of elements in them. For a K-hyperforest with n vertices, these elements are the $O(2^K n)$ supervertices.

The running time of m calls to INSERT is dominated by one Union operation for each of the $O(2^K)$ supervertices of h_{new}. Since operations on S_k require $O(k)$ expected time and m operations on a Union-Find structure require $O(\alpha(m,n))$ amortized time, the total running time of m INSERT calls is $O(2^K(K + \alpha(m,n)))$ (expected amortized).

For QUERY, computing $\{Z_k\}$ requires one INSERT call for each of the $O(2^K)$ k-supervertices of h_{new}, yielding a total time of $O(4^K(K + \alpha(m2^K, K)))$ time. Then, for each k and each pair of k-supervertices ($O(4^K)$ pairs in total), we must perform two hash table lookups and two Find operations. Thus, the total running time of m QUERY calls is $O(4^K(K + \alpha(m,n)))$ (expected amortized).

4.3 Correctness

Theorem 11. QUERY*(h_{new}) returns* TRUE *if and only if $H \cup \{h_{\text{new}}\}$ is acyclic, where H is the current hyperforest represented by $\{U_k\}$.*

Proof. We will show that QUERY(h_{new}) returns FALSE iff $H \cup \{h_{\text{new}}\}$ contains a hypercycle, the certificate of cyclicity that we developed in Section 3.2.

To do this, we consider the *hierarchical incidence graph structure* of H, which is the nested set of incidence graphs where each superedge node contains the incidence graph of that superedge, and the outermost incidence graph is the singleton node, the -1-superedge H; Figure 4(c) shows an example. All of the nested incidence graph structures are simply acyclic. For the purposes of the proof, we will

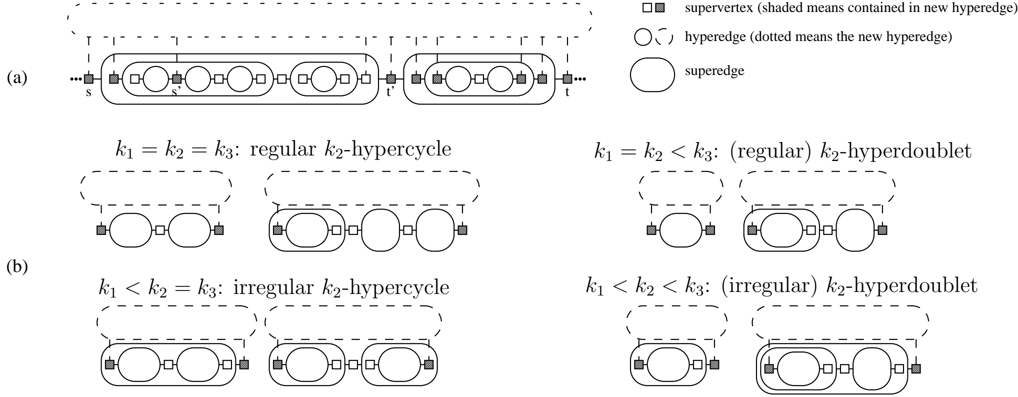


Figure 7: Illustration of the proof. (a) shows an example of a path through the current hyperforest H from supervertices s to t in relation to h_{new}, and one possibility for s', t'. (b) details the four different cases, where k_1, k_2, k_3 are the sizes of the smallest three overlaps on the path between supervertices s' and t'. For each case, two examples are shown. Note that all overlaps on the same level have the same size, while the overlaps one level deeper are one size larger.

modify these incidence graphs slightly: for each k-superedge, we add all k-supervertices to the incidence graph as overlap nodes. The resulting graph is still simply acyclic since only leaves have been added to the incidence graph.

A *hierarchical path* through this hierarchical incidence graph structure is a sequence of superedges and supervertices (Figure 7(a)). Compared with a standard path, a hierarchical path expands each k-superedge into a nested hierarchical path between two $k+1$-supervertices; note that supervertices may be adjacent in a hierarchical path. We construct such a path as follows: Consider any two distinct k-supervertices s and t connected to the same k-superedge p. Let us focus on T_p, the incidence graph structure of p, whose nodes are $k+1$-superedges and $k+1$-overlaps. Since T_p is simply acyclic, we can pick any closest pair of $k+1$-overlaps s' and t' that contain s and t respectively (s' might be the same as t' if it contains both s and t). After finding a path from $s' \supset s$ to $t' \supset t$ through T_p, we can recursively find paths through each of the $k+1$-superedges on this path, and concatenate all the pieces to form the final hierarchical path (Figure 7(a)).

$\Longrightarrow$: Assume QUERY(h_{new}) return false, and let s, t be the two k-supervertices that caused the test on line 8 to fail. Consider a hierarchical path P from s to t as described above. All overlaps along this path are of size at least k since s and t are k-reachable in H ($U_k(s,t)$ holds). Consider a minimal length sub-path $P' = s', \ldots, t'$ such that s' and t' are contained in h_{new}, and no overlap in P' is contained in h_{new}. Since s and t are not k-reachable in $H(h_{\text{new}})$ ($Z_k(s,t)$ holds), such a sub-path must exist.

Consider the sizes $k_1 \leq k_2 \leq k_3$ of the smallest three overlaps on the hierarchical path P'. We will now show that there is a k_2-hypercycle composed of h_{new} and the path from s' to t'. Intuitively, these three overlaps determine the partitioning into k_2-superedges and hence the type of hypercycle. Superedges spanning larger overlaps will be clumped into the same k_2-superedge. Figure 7(b) shows the various cases.

Let us construct a mixed sequence $h_{\text{new}}, s', q_1, s_1, \ldots, q_m, t', h_{\text{new}}$ of supervertices, superedges, and the hyperedge h_{new} by traversing P' path from s' to t' as follows: start from s'; at an overlap u, if $|u| + 1 \leq k_2$, step into the superedge after u, arriving in a strict superset of u; but if $|u| = k_2$, step over the entire superedge, arriving at another supervertex of the same size as u; repeat until we reach t'. Replace two adjacent overlaps with the smaller one.

We will now show that the superedges $q_1, \ldots, q_m$, and perhaps also h_{new}, form a k_2-hypercycle. If $|s'| = |t'| = k_2$, then the k_2-hypercycle includes h_{new} as a seperate k_2-superedge between q_m and q_1. Otherwise, h_{new} is merged with some q_i. We show two properties: First, the overlaps in this mixed

sequence are representative of the "maximum interaction" between either $h_{\rm new}$ and a k_2-superedge q_i or between two k_2-superedges q_i, q_j, because only the overlaps s' and t' (and not other overlaps in P') are contained in $h_{\rm new}$, and the overlaps were representative before adding $h_{\rm new}$. In other words, we know that two k-superedges would end up in the same k-superedge in the new hyperforest $H \cup \{h_{\rm new}\}$ iff there is a path through $>k$-overlaps along the (circular) sequence. Second, because we have constructed the hierarchical path through a k-superedge by taking the closest pair of $k+1$-supervertices, no overlap in our mixed sequence is a strict subset of another overlap. From these two properties, representative interactions and distinct overlaps, it follows that $q_1, \ldots, q_m, h$ form a hypercycle (see Figure 7(b) for examples of the various types of hypercycles).

$\Longleftarrow$: To show the inverse, assume that QUERY($h_{\rm new}$) returns true, which means every hierarchical path between any two k-supervertices s, t is entirely contained in $h_{\rm new}$. We will now show the incidence graph structures of $H \cup \{h_{\rm new}\}$ are simply acyclic. Consider all k-superedges that overlap at least $k$with $h_{\rm new}$. Let T be the forest corresponding to the incidence graph structures over the $k-1$-superedges that contain this set of k-superedges. These k-superedges in T will belong with $h_{\rm new}$ in the same $k-1$-superedge p in the new incidence graph structure T' of p. The k-superedges in T that have $>k$-overlap with $h_{\rm new}$will merge into the same k-superedge as $h_{\rm new}$, and in T', there will be an edge between a k-superedge and the k-superedge of $h_{\rm new}$ if the two exactly k-overlap.

We now show that this T' is simply acylic. To see that there are no cycles, notice that for any path of k-superedges $q_1, \ldots, q_m$ in T with q_1, q_m k-overlapping with $h_{\rm new}$, every k-superedge q_i for $1 < i < m$ is contained entirely in $h_{\rm new}$ and thus merges with $h_{\rm new}$. The end result is that $h_{\rm new}$ will merge with a set of connected components in T (possibly none). Thus, no cycles are created. Since all the new incidence graph structures are simply acyclic, $H \cup \{h_{\rm new}\}$ is acyclic. □

5 Experiments

We consider an application of our data structure: constructing a high-weight K-hyperforest given a weighted hypergraph. This problem is NP-hard for $K > 1$ [Sre00]. A common greedy heuristic for constructing a high-weight K-hyperforest is Prim-like [Mal91]: start with the highest-weight hyperedge, and iteratively add hyperedges, but at each iteration considering only candidate hyperedges that contain exactly Kvertices already in the current hyperforest and one new vertex. The maximum weight such hyperedge is added to the hyperforest. We suggest an alternative Kruskal-like greedy procedure, where at each iteration all hyperedges preserving acyclicity are considered. This might enable us to make better (greedy) choices, allows us to end up with a non-maximal hyperforest if some weights are negative, and allows us to seed the greedy procedure from an arbitrary hyperforest.

We conducted experiments to demonstrate the advantage of the less-limited greedy choices of the Kruskal-like procedure over the Prim-like procedure. We generated random weights on all candidate 2-hyperedges in a hypergraph with 100 vertices in the following way: we first constructed a random "planted" 2-hypertree by augmenting a hyperforest randomly. Hyperedges outside the planted hypertree were assigned a random weight uniformly distributed between 0 and 1. In one set of experiments, weights inside the hypertree were assigned random weights uniformly distributed between 0 and 10. In the second set, the weights were chosen uniformly between 0 and 1 with probability $1/2$, and between 0 and 20 with probability $1/2$. We generated 10 random weight-sets of each type, and tried both greedy approaches on each graph. Table 1 summarizes the weights of the resulting hypertrees. Kruskal performed significantly better on both sets of experiments, and especially when the weight was less evenly distributed in the "planted" hypertree.

Next, we demonstrate that out proposed data structure is practical and provides a substantial speed-up over existing methods based on Graham reductions to test for acyclicity. A Graham reduction is found by iteratively removing a leaf vertices (vertices that are incident to only one maximal hyperedge). For tree-widths 2 and 3, we construct a complete k-hypergraph with 25–100 vertices and uniformly

	$U[0,10]$	$\frac{1}{2}U[0,1]+\frac{1}{2}U[0,20]$
Planted	0.590 ± 0.0339	0.609 ± 0.059
Prim-like	0.506 ± 0.0816	0.323 ± 0.107
Kruskal-like	0.587 ± 0.0342	0.619 ± 0.058

Table 1: Averages and standard deviations of fraction of the weight captured by the hypertrees.

	$K=2, N=25$	$K=2, N=50$	$K=2, N=100$	$K=3, N=25$	$K=3, N=50$
Graham reduction	6.02s ± 1.14	3.61m ± 0.42	2.45h ± 0.09	34.90s ± 2.86	46.62m ± 5.27
Data structure	2.04s ± 0.33	36.27s ± 4.49	11.52m ± 0.58	21.78s ± 2.30	12.51m ± 1.58
Speedup factor	2.96 ± 0.36	5.98 ± 0.28	12.82 ± 1.17	1.61 ± 0.10	3.73 ± 0.08

Table 2: Averages and standard deviations over fi ve trials of the total execution times and speedup factor for a Kruskal-like hyperforest constructing algorithm using Graham reduction or our data structure to check acyclicity. N is the number of vertices, and K is the tree-width. The experiments were conducted on a 2.8 GHz P4.

random weights on all maximal and non-maxial hyperedges. Table 2 compares the running time of a Kruskal-like procedure for computing the maximum hyperforest using both a Graham reduction and our proposed data structure to test for acyclicity.

6 Discussion

We have presented a dynamic data structure for keeping track of acyclicity in hypergraphs and demonstrated its utility in a possible application: a Kruskal-like greedy approach for fi nding maximum weight acyclic hypergraphs. We have also developed two new ways of looking at hyperforests, via a hierarchical decomposition into superedges and via the lack of hypercycles. Beyond their utility in our algorithms and proofs, these new characterizations may be useful in analyzing acyclicity in other contexts.

We note that the data structure we present can easily be modifi ed to allow hyperedge deletions by using a variant of the Union-Find data structure that supports deletions (e.g. [Tho99], replacing the inverse Ackerman dependence on n with a $O(\log n / \log\log\log n)$ dependence for queries and $O(\log n(\log\log n)^3)$ for insertions and deletions).

It may be possible to improve the $O(4^K)$ factor in the running time of the operations of our data structure, since there is redundant work in the current QUERY and INSERT operations (although we cannot hope to completely avoid the exponential in K). For instance, observe that if s and t are k-reachable then any $s' \subset s$ and $t' \subset t$ are also reachable. Also, we might not need to check $O(n^2)$ pairs from a set of n supervertices if all are reachable.

Acknowledgments We are thankful to David Karger for guidance and suggestions about presentation, and to Erik Demaine for comments.

References

[Bes74] Julian Besag. Spatial interaction and the statistical analysis of lattice systems. *Proceedings of the Royal Statistical Society, Series B*, pages 192–236, 1974.

[BFMY83] Catriel Beery, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *J of the ACM*, 30(3):479–513, 1983.

[BJ02] F. R. Bach and M. I. Jordan. Thin junction trees. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 569–576, Cambridge, MA, 2002. MIT Press.

[Bod96] Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.

[BP01] Jozsef Bokszar and Andras Prekopa. Probability bounds with cherry trees. *Mathematics of Operations Research*, 26(1):174–192, 2001.

[CL68] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, IT-14(3):462–467, 1968.

[Cou90] B. Courcelle. The monadic second-order logic of graphs i: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

[KS01] David Karger and Nathan Srebro. Learning Markov networks: Maximum bounded tree-width graphs. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms*, 2001.

[Mal91] Francesco M. Malvestuto. Approximating discrete probability distributions with decomposable models. *IEEE Transactions on Systems, Man and Cybernetics*, 21(5):1287–1294, 1991.

[Mat99] Nicholas Matsakis. Recognition of handwritten mathematical expressions. Master's thesis, Massachusetts Institute of Technology, 1999.

[SG97] Kirill Shoikhet and Dan Geiger. A practical algorithm for finding optimal triangulations. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 185–190, 1997.

[Sre00] Nathan Srebro. Maximum likelihood Markov networks: An algorithmic approach. Master's thesis, Massachusetts Institute of Technology, 2000.

[Sre01] Nathan Srebro. Maximum likelihood bounded tree-width markov networks. In *The 17th Conference on Uncertainty in Artificial Intelligence*, 2001.

[Tho99] Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, pages 343–350, 1999.

[Tom86] Ioan Tomescu. Hypertrees and bonferroni inequalities. *J. Combin. Theory Ser. B*, 41:209–217, 1986.

[Wor82] K J Worsley. An improved Bonferroni inequality and applications. *Biometrika*, 69:297–302, 1982.