A Dynamic Data Structure for Checking Hyperacyclicity

Percy Liang Nathan Srebro

MIT Computer Science and Artificial Intelligence Laboratory
Cambridge, MA 02139, USA
{pliang,nati}@mit.edu

Abstract

We present a dynamic data structure that keeps track of an acyclic hypergraph (equivalently, a triangulated graph) and enables verifying that adding a candidate hyperedge (clique) will not break the acyclicity of the augmented hypergraph. This is a generalization of the use of Tarjan's Union-Find data structure for maintaining acyclicity when augmenting forests, and the amortized time per operation has a similar almost-constant dependence on the size of the hypergraph. Such a data structure is useful when augmenting acyclic hypergraphs, e.g.˜in order to greedily construct a high-weight acyclic hypergraph. In designing this data structure, we introduce a hierarchical decomposition of acyclic hypergraphs that aid in understanding *hyper-connectivity*, and introduce a novel concept of a *hypercycle* which is excluded from acyclic hypergraphs.

1 Introduction

Acyclic hypergraphs, or *hyperforests* (such as the one in Figure 1(a)), 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 those 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 desired.

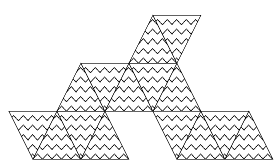

Figure 1: An example of a 2-hyperforest.

Acyclicity can allow 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 such applications the computation is often exponential in the *width* of the hypergraph, which corresponds to the maximum size of the hyperedges (or cliques in a triangulated graph). The class of K-hyperforests (width at most K) is then of particular interest.

When a K-hyperforest is necessary, one often wishes to choose the *best* possible K-hyperforest, where the quality of a hyperforest is captured as the sum of precomputed weights over its hyperedges, leading to the problem of finding a maximum-weight K-hyperforest [KS01]. Such a procedure is common in several different domains, for the special case where $K = 1$ and one seeks a maximum-weight tree: maximum likelihood Markov trees, known as Chow-Liu trees [CL68]; Hunter-Worsley trees for Bonferroni inequalities [Wor82]; and when trees are used to ensure efficient combinatorial optimization e.g. [Mat99]. Generalizations to higher width hyperforests are possible, and 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. The common heuristic approach is a Prim-like greedy approach, maintaining a *fully connected hypertree*, and adding to it only single vertices [Mal91, BP01, BJ02]. Alternatively, one might consider a more flexible, and possibly more powerful, Kruskal-like greedy approach, adding hyperedges to a possibly unconnected K-hyperforest. In order to do so, it is necessary to ensure that a new hyperedge about to be added does not break the acyclicity.

A particular situation where the Kruskal-like approach is necessary is when we would like to greedily augment an initial, possibly unconnected, hyperforest. This might be a required, or strongly desirable, substructure, or 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]).

Acyclicity is also important in order to preclude possible conflicts in, e.g. relational databases [BFMY83], or when learning graphical models [Bes74]. In such applications, one might want to ensure that new relations added, e.g. to a database scheme, do not break its acyclicity.

When augmenting forests, Tarjan's Union-Find dynamic data structure enables checking efficiently if a new edge breaks the acyclicity, by keeping track of the connected components in the graph. The main result in this paper is a dynamic data structure that serves a purpose analogous to Tarjan's Union-Find structure in hyperforests: for any candidate hyperedge, the data structure enables verifying that adding the hyperedge will not break the acyclicity of the augmented hyperforest. The amortized time per operation is almost independent of the hypergraph size (dependent through the inverse of Ackarman's function).

We show how in hyperforests, it is no longer enough to consider a single type of connectedness. Thus, the simple notion of connected components, which can be captured using a single Union-Find structure, is not enough. Instead, we present a novel view of hyperforests, at different levels, each highlighting a different degree of connectivity, and use a separate Union-Find structure for each level.

On the way to developing such a data structure, we also suggest the notion of a *hypercycle*. Although acyclic hypergraphs have been studied for the past three decades, using many equivalent characterizations, we are not aware of any characterization that directly defines a *hypercycle* and characterizes acyclic hypergraphs as those that to do not have hypercycles. Such a characterization, which we give in Definition 9 and Theorem 12, provides added insight into hyperforests.

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 examine the concepts of hyperconnectivity and hypercycles and lay the foundations for the proof of the data structure, which is presented in Section 4. Finally, in Section 5 we discuss the problem of finding maximum weight hypertrees and the utility of Kruskal-like greedy approaches over Prim-like approaches.

2 Hypergraph Acyclicity

Preliminaries A *hypergraph* $H(V)$ is a collection of subsets, or *hyperedges*, of the vertex set V: $H(V) \subset 2^V$. If $h' \subset h \in H$ then the hyperedge h' is *covered* by H. Of particular interest are the *maximal hyperedges* of a hypergraph H, which are not covered by any other hyperedges in H—in fact, in this paper we refer to H as containing only such maximal hyperedges, while denoting by $\overline{H}$ the collection of all covered hyperedges: $\overline{H} = \{h \subset V | \exists_{h' \in H} h \subseteq h'\}$. We say that a hypergraph H_1 covers H_2 if $H_2 \subset \overline{H_1}$.

The *projection* of a hypergraph H onto a set of vertices s is $H_s = \{h \cap s | h \in H\}$.

Several equivalent definitions of hypergraph acyclicity are in common use (see [Sre00] for a review). Here, we define acyclicity using the notion of a tree structure:

Definition 1. *A hypergraph H is said to have a* tree structure $T(H)$ *iff T is a tree over all the hyperedges of H and the following* path overlap property *holds: If $(h_1, h_2, \ldots, h_m)$ is a path of H-hyperedges in T, then* $\forall_{1<i<m}\, h_1 \cap h_m \subseteq h_i$.

Definition 2. *A hypergraph is* acyclic *iff it has a tree structure. An acyclic hypergraph is also referred to as a* hyperforest. *We say that a hyperforest has width (at most) K, and refer to it as a K-hyperforest, if its hyperedges are of size at most $K + 1$.*

Problem Statement: Checking Acyclicity We seek a data structure that will allow us to augment a hyperforest by adding hyperedges to it, ensuring that it remains acyclic. That is, the data structure should keep track of the "current" hyperforest H and support two operations, where h_{new} is a hyperedge:

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 it is acyclic.

Bounded Tree-Width Hypergraphs Unlike forests, hyperforests do not form a monotone family of hypergraphs: a sub-hypergraph of an acyclic hypergraph might be cyclic, and conversely, adding hyperedges to a cyclic hypergraph might make it acyclic. When a tree structure is used in order to perform efficient computation using dynamic programming (e.g.̃inference in graphical models), it is often admissible to add extra hyperedges to a cyclic hypergraph in order to obtain a covering hyperforest. Computation is then performed using the tree structure of this covering hyperforest. The important requirement is that the width of the covering hyperforest be small, as computation is exponential in this width:

Definition 3. *The* tree-width *of a hypergraph H is the minimum width of a hyperforest that covers H.*

Accordingly, in such situations, one might want a have data structure that checks whether adding a hyperedge maintains low tree-width. If all hyperedges added are of size at most $K + 1$, then the data structure presented here ensures a tree-width of not more than K. The converse is not true: a hyperedge h_{new} might be refused even though $H \cup \{h_{\text{new}}\}$ has tree-width at most K.

Before considering dynamic data structures for maintaining low tree-width, it is important to remember that calculating the tree-width statically, or equivalently finding a narrow triangulation, is by itself a very difficult task. Although linear time algorithms for constant width have recently been discovered [Bod96], the dependence on the 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].

Augmentation in Greedy Algorithms Our main motivation for the dynamic data structure stems from greedy Kruskal-like construction of high-weight hyperforests, particularly in order to find maximum likelihood Markov networks of bounded tree-width. If the weights on hyperedges are all non-negative, it may be appropriate to allow bounded tree-width hypergraphs in intermediate stages, requiring a dynamic data structure that checks tree-width rather then acyclicity. However, in situations in which weights might be negative or positive, as in the case when the weight of a hypergraph corresponds to its likelihood [Sre01], we cannot allow intermediate cyclic hypergraphs, as making them acyclic might introduce high negative weights (the weight of a cyclic hypergraph does *not* correspond to its likelihood). For such applications, the acyclicity is the correct property to require (along with ensuring each added hyperedge is of proper size).

3 A new look at hyperforests

3.1 Hyperconnectivity

Connectivity in hyperforests is more substantially complex than connectivity in forests. In forests, two edges are either incident or disjoint. In a K-hyperforest, there are $K+1$ "degrees" at which two hyperedges can overlap, corresponding to overlap sizes ranging from 0 to K. We suggest a hierarchical decomposition of a hyperforest into *superedges*, which allows us to concentrate on one degree of connectivity at a time.

Definition 4 (Superedge). *Two hyperedges h_1 and h_m are k-connected*[1] *if there exists a sequence of hyperedges $h_1, \ldots, h_m$ such that $|h_i \cap h_{i+1}| \geq k$ for all $1 \leq i \leq m-1$. A k-*superedge *q of a hyperforest H is a maximal $(k+1)$-connected subset of H.*

Denote the set of all vertices in the union of all the hyperedges in a superedge q as $\tilde{q} = \cup q$.

Definition 5 (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 $s = h_1 \cap h_2$ for some $h_1 \in q_1$, $h_2 \in q_2$.*

Figure 2 shows a hierarchical decomposition of a 3-hyperforest into superedges. Each $(k-1)$-superedge contains a tree structure over the k-superedges that is a minor of the tree structure over all the hyperedges. In a K-hyperforest H, the K-superedges are the hyperedges, the 0-superedges are the connected components, and the single (-1)-superedge is the entire hyperforest H.

We can now look at a K-hyperforest one *level* at a time. At level k, we consider only k-connectivity by focusing on the k-superedges in a $(k-1)$-superedge. $(< k)$-connectivity does not exist in a $(k-1)$-superedge[2], and $(> k)$-connectivity is abstracted into the k-superedges. As a result, the tree structure over the k-superedges in a $(k-1)$-superedge has the desirable property that all overlaps between adjacent k-superedges have the same size.

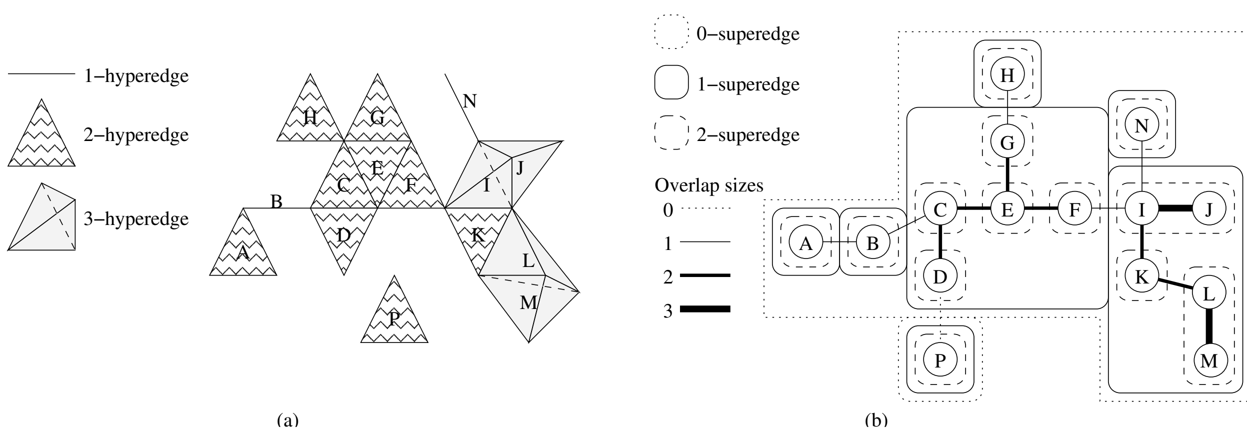

Figure 2: An example of the hierarchical decomposition of superedges. (a) depicts a hypergraph H, and (b) shows the tree structures of H at different levels.

Lemma 6. *The path in a tree structure T between two hyperedges in the same superedge q contains only hyperedges from q (i.e. the hyperedges of a superedge appear contiguously in T).*

[1] Note that this is not the usual definition of k-connected.

[2] In this way, a k-superedge q 'functions' as a k-hyperedge because all overlaps between q and any hyperedge not in q have size at most k.

Proof. Since any overlap along the path in T between two hyperedges is a separator between them, and the two hyperedges are $(k+1)$-connected, all overlaps must be of size at least $k+1$, so all hyperedges along the path are $(k+1)$-connected. □

Let us now show that a hierarchical decomposition such as the one in Figure 2 exists for all hyperforests. We do this by constructing, for each $(k-1)$-superedge p, a tree structure over the k-superedges of p.

Definition 7. *If p is a $(k-1)$-superedge, a tree structure T_Q over a set of k-superedges Q_p in p is a tree such that the following hold:*

1. *Any two k-superedges in Q_p overlap simply.*
2. *For all paths $q_1, \ldots, q_m$ in $T_Q, \forall 1 \leq i \leq m, \tilde{q}_1 \cap \tilde{q}_m \subset \tilde{q}_i$.*

Theorem 8. *A hypergraph H is a hyperforest iff for all k, for all $(k-1)$-superedges p, there exists a tree structure over the set Q_p of k-superedges of p.*

Proof.

$\Longrightarrow$: From the tree structure T_H of H, we will construct a tree structure T_Q over Q_p, which is a minor of T_H. Include $(q_1, q_2) \in T_Q$ iff there is some edge $(h_1, h_2) \in T_H$ with $h_1 \in q_1$ and $h_2 \in q_2$. Call h_1 and h_2 the *gateway hyperedges* of the edge $(q_1, q_2) \in T_Q$. Now we have to show that T_Q is a valid tree structure.

To verify that T_Q is actually a tree, we will show that a cycle $q_0, q_1, \ldots, q_m$ in T_Q means there is a cycle in T_H.[3] Let $h_{i-1} \in q_{i-1}$ and $g_i \in q_i$ be the two gateway hyperedges of (q_{i-1}, q_i). By Lemma 6, the path from g_i to h_i in T_H contains only hyperedges in q_i. Consider the sequence $c = h_0, g_1, \ldots, h_1, g_2, \ldots, h_2, \ldots, g_m, \ldots h_m$. In any case, there are at least $m \geq 3$ distinct hyperedges, so c is a cycle in T_H.

To verify that all superedges overlap simply, we will show that the overlap between $q_1, q_m \in Q$ is contained in gateway hyperedges of q_1 and q_m. Let $q_1, \ldots, q_m$ be the path in T_Q, and $h_1 \in q_1$ and $h_m \in q_m$ be gateway hyperedges of (q_1, q_2) and (q_{m-1}, q_m), respectively. Both h_1 and h_m must be on the path in T_H from any $h'_1 \in q_1$ to any $h'_m \in q_m$. Because T_H is a tree structure, $h'_1 \cap h'_m \subset h_1 \cap h_m$, so $\tilde{q}_1 \cap \tilde{q}_m \subset h_1 \cap h_m$.

To verify the path overlap property, we invoke the path overlap property of T_H and notice that the overlaps along the path between two k-superedges $q_1, q_m \in Q$ are included in the overlaps along a corresponding path in T_H. Let s be the overlap between q_1 and q_m ($s = h_1 \cap h_m$ for some $h_1 \in q_1$, $h_m \in q_m$). The path from h_1 to h_m in T_H contains the gateway hyperedges of each (q_i, q_{i+1}). Since s is contained in every hyperedge in the path $h_1, \ldots, h_m$ (in particular, the gateway hyperedges), s is also contained in every superedge in the path $q_1, \ldots, q_m$.

$\Longleftarrow$: To show that H is a hyperforest, we will construct a tree structure T_H over H. To do this, we assume by induction that there is a tree structure T_q over each $(k+1)$-superedge T_q of H and then show that there is a tree structure T_p over each k-superedge p of H. When $k = -1$, $p = H$ is the single (-1)-superedge, and we have the desired tree structure $T_H = T_p$.

Base case ($k = K$): Each k-superedge is a single hyperedge and has a trivial tree structure.

Inductive case ($k < K$): Assume that there is a tree structure T_q over each $(k+1)$-superedge q of H. Fix any k-superedge p. p is partitioned into a set Q of $(k+1)$-superedges, each of which has a tree structure by the induction hypothesis. Furthermore, there is a tree structure T_Q over Q. We now construct T_p. T_p includes $\cup_{q \in Q} T_q$ plus an edge (h_{q_1}, h_{q_2}) for each $(q_1, q_2) \in T_Q$, where $h_{q_1} \in q_1$ and $h_{q_2} \in q_2$ are chosen to be any hyperedges containing $\tilde{q}_1 \cap \tilde{q}_2$. It is clear that T_p is a valid tree structure.

□

[3]Recall that a (simple) cycle is a closed path of at least three vertices, all distinct. To clarify notation, if we say $q_0, q_1, \ldots, q_m$ is a cycle, $q_0 = q_m$ and $m \geq 3$.

3.2 Hypercycles

A tree structure proves the acyclicity of a hypergraph, whereas a hypercycle proves the non-acyclicity of a hypergraph.

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

1. *Two k-superedges that overlap non-simply. Call this a* hyperdoublet*.*
2. *A sequence $c = q_1, q_2, \ldots, q_m$ ($m \geq 3$) of distinct k-superedges with distinct overlaps $s_i = \tilde{q}_i \cap \tilde{q}_{i+1}$ of size exactly k, such that there exists some s_i that does not contain $s_* = \tilde{q}_1 \cap \tilde{q}_m$. If $|s_*| = k$, call c a* regular hypercycle*. Otherwise, call c a* hyperloop*.*

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_* is the vertex between the "first" and "last" edge. In higher order cycles, we cannot always require that the overlap s_* that "closes" the cycle is also of size exactly k, e.g. (c) in Figure 3. However, it is not enough to require that the overlap s_* be non-empty (e.g. (d) in the Figure): to be a cycle a path much "touch" itself "outside" of the common overlap in the path.

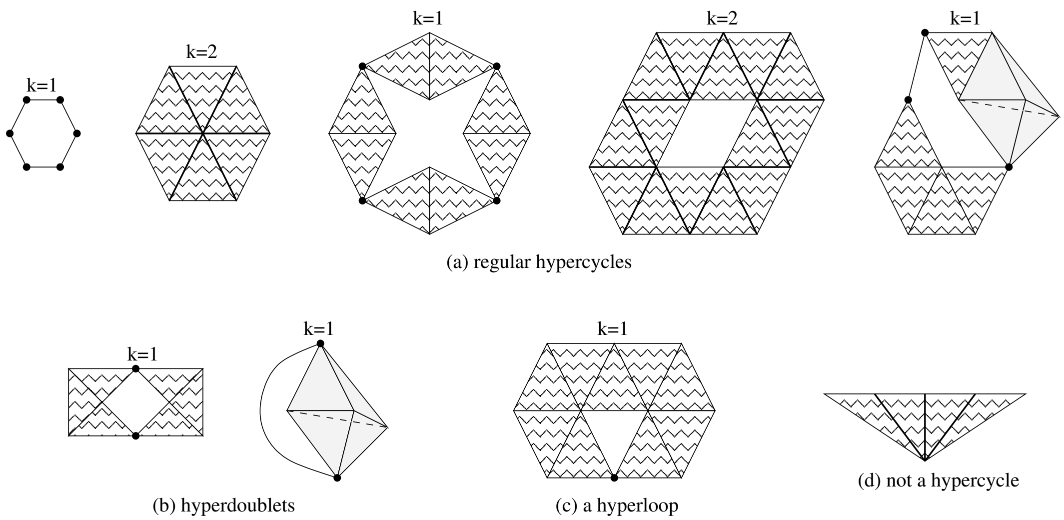

Figure 3: Examples of hypercycles.

Before introducing the main theorem of this section, we state the following lemma which facilitates discussing hypergraphs without hypercycles:

Definition 10. *We say that a sequence of overlaps $s_1, \ldots, s_m$ is* block-distinct *if all repeated overlaps occur next to each other (if $s_i = s_j$, then $\forall i \leq k \leq j, s_i = s_k = s_j$).*

Lemma 11. *A hypergraph has no k-hypercycles iff both of the following conditions are true:*

1. *All k-superedges overlap simply.*
2. *For all sequences $q_1, q_2, \ldots, q_m$ ($m \geq 3$) of block-distinct k-superedges with distinct overlaps $s_i = \tilde{q}_i \cap \tilde{q}_{i+1}$ of size exactly k, $s_* = \tilde{q}_1 \cap \tilde{q}_m$ is contained in every s_i.*

Proof. The difference between this lemma and the definition of a hypercycle is that in condition 2, we require block-distinct, whereas in part 2 of Definition 9, we say distinct. Before we show that distinct and block-distinct are interchangeable, we note that in condition 2 of Lemma 11, $\forall i, s_* \subset s_i$ can be equivalently replaced by $\exists i, s_* \subset q_i$: suppose that $s_* \subset q_i$ for some i. Then $s_* \subset q_1 \cap q_i$ and $s_* \subset q_i \cap q_m$. By induction on the two halves of $q_1, \ldots, q_m$, $s_* \subset q_i \forall i$.

Now, for each sequence $c = q_1, \dots, q_m$ with block-distinct overlaps, we construct a subsequence c' with distinct overlaps by replacing each maximal contiguous subsequence $q_i, \dots, q_j$ in c with identical overlaps along $q_i, \dots, q_j$ with just q_i and q_j. The overlaps along c' are distinct and also contain s_*. □

Requiring lack of hypercycles resembles the definition of a tree structure over superedges (Definition 7). The main difference between the two conditions is that the path overlap property requires agreeing on a global object, namely the tree structure, and only paths along along the tree structure must have the path overlap property. The independence from a specific tree structure, which will be crucial later in proving the correctness of the data structure, is achieved by requiring a uniform degree of connectivity between superedges.

Theorem 12. *H is a hyperforest iff H contains no hypercycle.*

Proof.
$\Longrightarrow$: Let T_H be any tree structure of H. We will verify the two conditions in Lemma 11.

1. To show that any two k-superedges q_1, q_2 overlap simply, let k' be the largest value such that q_1 and q_2 are in the same $(k'-1)$-superedge p. Let r_1 be the k'-superedge that contains q_1, and $r_2 \neq r_1$ be the k'-superedge that contains q_2. By Theorem 8, r_1 and r_2 overlap simply, so q_1 and q_2 also overlap simply.

2. Let $c = q_1, \dots, q_m$ be a sequence of distinct k-superedges with distinct overlaps s_i of size k. We want to show that for all i, $s_* \subset s_i$. Consider the path c' in T_Q which is the concatination of the paths in T_Q between every q_i, q_{i+1}: $c' = q_1, \dots, q_2, \dots, q_m$. Since all overlaps s_i are of size k, all adjacent k-superedges in c' in the sub-path $q_i, \dots, q_{i+1}$ have the same overlap s_i. Label each edge (q, r) along the path c' in T_Q, with the overlap $\tilde{q} \cap \tilde{r}$. Then, for $i \neq j$, the set of edges from q_i to q_{i+1} and the set of edges from q_j to q_{j+1} are disjoint because their labels are different ($s_i \neq s_j$). Of course, for all i, the edges between q_i and q_{i+1} are distinct because they form a simple path. Therefore, all edges in the path c' are distinct, and c' is a simple path. Moreover, T_Q is a tree structure, so $\forall 1 \leq i \leq m, s_* \subset \tilde{q}_i$.

$\Longleftarrow$: For a hypergraph H with no hypercycles, we will show that H is a hyperforest by constructing a tree structure T_p over the set of k-superedges Q_p of each $(k-1)$-superedge p (Theorem 8). For each covered hyperedge (subset of a hyperedge) s of size exactly k, consider the k-superedges $q_1, q_2, \dots, q_m \in Q_p$ that contain it. Note that since there are $(k-1)$-superedges in a k-superedge, their overlap is of size exactly k, and is therefore exactly s. Choose one of these, say q_1, arbitrarily and connect all remaining $q_2, \dots, q_m$ to q_1 in T_p (i.e. $\forall_{1<i\leq m}(q_1, q_i) \in T_p$).

To show that T_p is a tree, label each edge in T_p with its corresponding overlap. By construction of T_p, for each label, there is a single k-superedge (q_1 in the above notation) that is incident to all edges so labeled. Therefore, every simple path must contain at most two edges of the same label, and if there are two such edges in the path, they must be adjacent. Suppose for contradiction that there is a simple cycle $c = q_0, q_1, \dots, q_m$ in T_p with $s_i = \tilde{q}_i \cap \tilde{q}_{i+1}$ for $0 \leq i < m$, such that $s_0 \neq s_1$. $q_1, \dots, q_m$ is a sequence of distinct k-superedges with block-distinct overlaps. Due to condition 2 of Lemma 11, $s_0 \subset s_1$. But this is a contradiction since both overlaps are of size k. Thus, c could not have existed, and T_p is indeed a tree.

The simplicity of the overlaps and the path overlap property in T_p now follow immediately from the definition of a hypercycle.

□

QUERY(h_{new}):
compute $\overline{H_{h_{\text{new}}}}$ and Z_k
for $k \leftarrow K$ **downto** 1 **do**
  for $s, t \in S_k$ **do**
    if $U_k(s,t)$ **and not** $Z_k(s,t)$ **then**
      return FALSE
return TRUE

INSERT(h_{new}):
assert QUERY(h_{new})
for $k \leftarrow 1$ **to** K **do**
  for $s, t \in S_k$ **do**
    union (s,t) **in** U_k

Figure 4: Pseudocode for QUERY and INSERT. S_k denotes the set of k-supervertices in $H_{h_{\text{new}}}$.

4 The Data Structure

We use ordinary Union-Find structures at each of the K levels. At level k, a Union-Find structure U_k keeps track of disjoint sets of connected k-*supervertices*.

Definition 13 (Supervertex). *A k-*supervertex *(same as a covered $(k-1)$-hyperedge) of a hypergraph H is a k-subset of some hyperedge $h \in H$. Two k-supervertices s_1 and s_2 are k-connected (or just connected) if there exists a $(k-1)$-superedge q of H such that $s_1, s_2 \subset \tilde{q}$.*

4.1 Overview of the data structure state

We maintain the following two values with respect to the current hyperforest H:

- The set $\overline{H}$ of all covered hyperedges in H, equivalent to the supervertices of H.
- The transitive relation U_k for each $k \leq K$, where $U_k(s,t)$ specifies whether the two k-supervertices s and t are connected.

In addition, in each QUERY operation, we compute two additional values that are dependent on h_{new}. These two values are projections of $\overline{H}$ and U_k onto h_{new}.

- The set $\overline{H_{h_{\text{new}}}}$, which is the projection of $\overline{H}$ onto h_{new}.
- The relation Z_k for each $k \leq K$, where $Z_k(s,t)$ specifies whether two k-supervertices s and t are connected in $H_{h_{\text{new}}}$.

4.2 The QUERY and INSERT operations

QUERY(h_{new}) returns TRUE iff for some k, there exists two k-supervertices s and t such that s and t are connected in H but not connected in $H_{h_{\text{new}}}$. If QUERY(h_{new}) returns TRUE, INSERT(h_{new}) connects all k-supervertices of h_{new} in U_k. See Figure 4 for pseudocode.

Roughly speaking, QUERY returns FALSE when s and t are connected "outside" of h_{new}. In this case, adding h_{new} would close a hypercycle. It is not enough to merely require that s and t are connected in H, since if they are also connected to the same extent inside h_{new}, h_{new} might "collapse" into the path between s and t.

4.3 Correctness

We now show that QUERY(h_{new}) returns TRUE iff $H' = H \cup \{h_{\text{new}}\}$ is a hyperforest[4]. Let $\overline{H'} = \overline{H} \cup \{h_{\text{new}}\}$ be the supervertices of the augmented hyperforest.

Theorem 14. *If* INSERT(h_{new}) *returns* TRUE*, then H' is a hyperforest.*

Proof. From any tree structure T_H of H, we will construct a tree structure $T_{H'}$ for H'. The idea is to break up T_H into the subtrees $T_1(H_1), \ldots, T_m(H_m)$ that h_{new} separates, and then connect each subtree to h_{new} via some $h_i \in T_i$.

Specifically, to get $T_{H'}$, remove each edge (h_a, h_b) in T_H such that $h_a \cap h_b \subset h_{\text{new}}$. We claim that letting $h_i = \text{argmax}_{h' \in H_i} |h' \cap h_{\text{new}}|$ (the hyperedge that overlaps with h_{new} the most) creates a valid tree structure. It is enough to verify that the path overlap property holds for paths in $T_{H'}$ involving h_{new}: both paths $h_a, \ldots, h_{\text{new}}, \ldots, h_b$ passing through h_{new} and paths in which h_{new} is an endpoint.

Paths passing through h_{new} connect h_a, h_b from different subtrees, and so the path from h_a to h_b in T_H must have contained a removed edge, and $h_a \cap h_b \subset h_{\text{new}}$.

For paths where h_{new} is an endpoint, we show the path overlap property by showing that the sequence of overlaps between h_{new} and every hyperedge along the path from h_a to h_{new} in $T_{H'}$ increases telescopically. To do this, we must argue that (1) the overlaps form a subset relation, and that (2) there are no "local minimum" overlaps where both the preceding and following overlaps are proper supersets of an overlap. Formally, we verify two properties:[5]

1. For any $(h_a, h_b) \in T_H$, if $h_a \cap h_b \not\subset h_{\text{new}}$, then either $h'_a \subset h'_b$ or $h'_b \subset h'_a$. Otherwise, there would be two k-supervertices in h_a and h_b connected "outside of" H.
2. For any path h_a, h_b, h_c in T_H such that $h_a \cap h_b \not\subset h_{\text{new}}$ and $h_b \cap h_c \not\subset h_{\text{new}}$, then $|h'_b| \geq \min\{|h'_a|, |h'_c|\}$. Otherwise, there would be two k-supervertices in h_a and h_c connected "outside of" H.

□

Theorem 15. *If* QUERY(h_{new}) *returns* FALSE*, then H' is not a hyperforest.*

Proof. We will show how to construct a hypercycle using the two k-supervertices s, t that are k-connected "outside" h_{new}. We will construct a hypercycle that includes h_{new} and the k-connected path between s and t. Let $q_1, \ldots, q_m$ be the k-connected path in the tree structure T_Q over the k-superedges Q_p of the $(k-1)$-superedge p containing s and t.

Intuitively, $h, q_1, \ldots, q_m$ "forms a hypercycle," but h_{new} may collapse any of the q_i's into one k-superedge. So we extract out a subpath of $q_1, \ldots, q_m$ such that h_{new} collapses at most the terminals q_1, q_m.

For each i, let u_i be a maximum overlap between h_{new} and a hyperedge in the k-supervertex q_i along the path. We also require $|u_i| \geq k$; otherwise, we say that u_i does not exist. We choose u_1 and u_m so that $u_1 \supset s$ and $u_m \supset t$. See Figure 5 for an illustration.

Extracting the subpath proceeds in two steps:

1. Choose a subpath for which all overlaps between adjacent k-superedges on this subpath are not contained in h_{new}. This is possible because s and t are not k-connected in $H_{h_{\text{new}}}$.
2. Choose a subpath $q_i, \ldots, q_j$ from the resulting subpath of the first step such that the u_r's do not exist for $i < r < j$, but u_i, u_j do exist.

[4] Due to space limitations we provide only proof outlines—see [LS03] for full proofs.

[5] For notational convenience, denote the overlap of h_{new} and a hyperedge as the hyperedge with an added prime. For instance, $h'_a = h_a \cap h_{\text{new}}$.

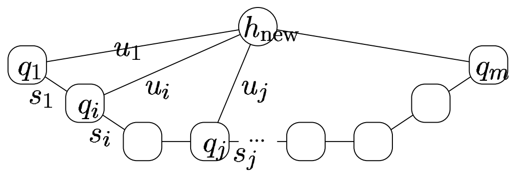

Figure 5: Construction of a hypercycle in Theorem 15.

Now, consider the k-superedges $c' = q'_1, \ldots, q'_l$ in H' after h_{new} optionally collapses the terminal superedges. All overlaps in this sequence are block-distinct and of size k. If $l' \geq 3$, c' is a regular hypercycle. If $l' = 2$, c' is a hyperdoublet. If $l' = 1$, we have to delve deeper and look at the k'-superedges of q'_1 for the smallest $k' > k$. There are at least two k'-superedges that result. If there are two, we have a hyperdoublet. Otherwise, we have a hyperloop around the overlap of size k.

Therefore, H' is not a hyperforest. □

4.4 Time complexity

In QUERY, we compute $\overline{H_{h_{\text{new}}}}$ by iterating through all subsets of h_{new} and selecting the ones that are in $\overline{H}$. We compute Z_k by iterating through all $h' \in H_{h_{\text{new}}}$ and unifying all pairs of k-subsets of h'. There are less than $4^{|h_{\text{new}}|}$ pairs of k-subsets in h_{new}, and so computing Z_k requires no more than $O(K4^K)$ time.

Each U_k can be implemented as an ordinary Union-Find structure. The number of supervertices stored in these Union-Find structures is bounded by the maximal number of covered hyperedges in a hyperforest of width K, which is less than $|V|2^{K+1}$. The amortized time of each call to Find is then $O(\alpha(|V|2^K)$, where α is the inverse Ackermann function. Each QUERY operations calls Find once for each pair of k-subsets in h_{new}, yielding a combined amortized run-time of $O(4^K(K + \alpha(|V|2^K)))$ per QUERY operation.

Each INSERT operation calls Union a similar number of times, its amortized run-time is also $O(4^K(K + \alpha(|V|2^K)))$.

5 Experiments with Greedy Hypertrees

Given as input weights on candidate hyperedges, the weight of a hyperforest is equal to the sum of the weights of all hyperedges it covers. In the K-maximum hypertree problem, we would like to find the K-hyperforest of maximum weight. When $K > 1$ the problem is NP-hard [Sre00]. Figure 6 provides an example where greedy approaches perform suboptimally.

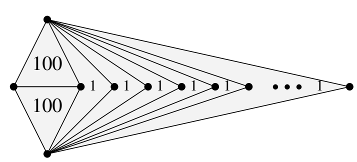

Figure 6: An example hypergraph on which a greedy algorithm would capture asymptotically none of the weight of the optimal hypertree.

The common greedy heuristic for constructing a high-weight hyperforest is Prim-like: we start with the highest-weight hyperedge, and at each iteration consider only candidate hyperedges of the form $s \cup \{v\}$, where $s \subset h \in H$ is a subset of size exactly K of a hyperedge of the "current" hyperforest, and v is a new vertex not yet in the hyperforest. The heaviest[6] such hyperedge is added to the hyperforest, which remains

[6] If weights are specifi ed also for non-maximal hyperedges, then when considering the weight of a hyperedge, the weights of all its sub-hyperedges not already covered by H are added to it.

fully K-connected.

Using the data structure described in this paper, one can consider a more flexible Kruskal-like greedy procedure: at each iteration, all hyperedges which do not cause hypercycles are considered, and the heaviest one is added to the hyperforest.

To demonstrate the possible utility of a Kruskal-like greedy procedure, as compared to a Prim-like greedy 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.

	$U[0,10]$	$\frac{1}{2}U[0,1]+\frac{1}{2}U[0,20]$
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: the planted hypertree, the hypertree recovered with a Prim-like greedy approach, and the one recovered with a Kruskal-like greedy approach.

6 Conclusion

We have presented a dynamic data structure for keeping track of acyclicity in hypergraphs, allowing augmenting a hyperforest while ensuring new hyperedges to not break its acyclicity. Each operation takes time which is almost constant in the size of the hyperforest but, like most hyperforest algorithms, is exponential in the tree-width. Although an exponential dependence is probably unavoidable, it might well be possible to reduce the precise dependence.

The new dynamic data structure allows efficient implementation of Kruskal-like greedy heuristics for finding high-weight hyperforests that have some advantages over Prim-like heuristics. However, the important problem of constructing efficient algorithms that approximate well the maximum-weight hypertree remains open.

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.

[LS03] Percy Liang and Nathan Srebro. A dynamic data structure for checking hyperacyclicity. Available on `theory.lcs.mit/˜naits/HyperTrees`, 2003.

[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.

[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.