How Much Of A Hypertree Can Be Captured By Windmills?

Percy Liang Nathan Srebro

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

Abstract

Current approximation algorithms for maximum weight *hypertrees* find heavy *windmill farms*, and are based on the fact that a constant ratio (for constant width k) of the weight of a k-hypertree can be captured by a k-windmill farm. However, the exact worst case ratio is not known and is only bounded to be between $1/(k+1)!$ and $1/(k+1)$. We investigate this worst case ratio by searching for weighted hypertrees that minimize the ratio of their weight that can be captured with a windmill farm. To do so, we use a novel approach in which a linear program is used to find "bad" inputs to a dynamic program.

1 Introduction

Acyclic hypergraphs, or *hyperforests*, 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.

Of particular interest are those hyperforests with hyperedges of bounded size: we refer to hyperforests with hyperedges of size at most $k+1$ as k-hyperforests, where 1-hyperforests are standard forests[1]. Given weights on hyperedges, one might seek to find the maximum weight k-hyperforest. Unlike the problem of finding the maximum weight forest, or 1-hyperforest, the problem of finding the maximum weight k-hyperforest is NP-hard [Sre00]. Karger and Srebro [KS01] presented a constant factor (for constant k) approximation algorithm for this problem.

In order to achieve a constant approximation ratio, Karger and Srebro introduced a hypergraph generalization of stars (trees of diameter at most 2), referred to as *windmills*, and observed that a constant fraction of the weight of a k-hyperforest can always be captured by a collection of disjoint k-windmills (a *windmill farm*). Unlike hyperforests, which can only be characterized by a global property (namely acyclicity), windmill farms can be characterized by local properties, enabling an integer programming formulation of the maximum k-windmill farm problem, that can be approximated to within a constant factor by rounding a relaxed linear program [KS01]. By the Windmill Cover Theorem [KS01], we know that a constant fraction of the weight of any k-hyperforest can be captured by a k-windmill farm. Combining the constant factor rounded linear program approximation algorithm for k-windmills with the Windmill Cover Theorem makes the rounded linear program a constant factor approximation algorithm for the maximum weight k-hypertree problem.

[1] k is known as the width.

The approximation ratio of such an algorithm depends on the *k-windmill coverage bound*, which is the fraction of the weight of a k-hyperforest that can always be captured by a k-windmill farm. The Windmill Cover Theorem establishes that the windmill coverage is at least $\frac{1}{(k+1)!}$; that is, for any weighted k-hyperforest, there always exists a k-windmill farm capturing at least $\frac{1}{(k+1)!}$ of its weight. However, a large gap remains between this lower bound and the worst known upper bound, $\frac{1}{k+1}$ [Sre00].

In the work reported here, we investigate the windmill coverage bound, by searching for "worst case" weighted hypertrees, that is, weighted k-hypertrees that minimize the ratio of their weight that can be captured by k-windmill farms. By finding such hypertrees, we are able to reduce the upper bound on the windmill coverage. Furthermore, by investigating the structure of the resulting hypertrees, we hope to be able to formulate and prove tighter general bounds on the windmill coverage.

In order to find such worst-case hypertrees, we present a novel general approach in which inputs to a dynamic program are encoded by a linear program, enabling us to find the "worst" input by solving the linear program. Specifically, we present algorithms for the following progression of problems:

Problem 1 Given a weighted k-hypertree, find the windmill coverage of the hypertree, i.e., the weight of the maximum weight k-windmill farm contained in it.

Problem 2 Given an unweighted hypertree, find the windmill coverage bound for that hypertree, namely the worst assignment of weights to its hyperedges such that the windmill coverage of the hypertree is minimized.

Problem 3 Given a width k, find the overall windmill coverage bound, namely the weighted k-hypertree with the minimum windmill coverage.

After defining the relevant combinatorial objects and concepts (Section 2), we present dynamic programming algorithm for Problem 1 (Section 3). Next, we formulate a linear program for Problem 2 (Section 4) that is based on the dynamic program, and encodes all possible inputs to it (i.e. all possible weight settings). For both of these problems, we first tackle the problem for $k = 1$, i.e. disjoint collections of stars (*skies*) in forests. Although skies are not necessary for finding maximum weight forests, and the worst case coverage ratio for skies is known and equal to two, studying the problem for $k = 1$ facilitates understanding of the problem for higher widths. In Section 5 we approach Problem 3 by applying the linear program for Problem 2 to larger and larger "complete" hypertrees.

Implementation Notes The methods described here were written in C++, compiled using GNU C++ compiler version 3.0, RedHat Linux 9.0. The LP solver used was CPLEX version 7.1.

2 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 the set of maximal hyperedges, while denoting by $\overline{H}$ the collection of all hyperedges: $\overline{H} = \{h \subset V | \exists_{h' \in H} h \subseteq h'\}$. We say that a hypergraph H_1 covers H_2 if $H_2 \subset \overline{H_1}$.

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 all its hyperedges are of size at most $k+1$.*

Definition 3. *Let $T(V)$ be a rooted tree with depth at most k. A k-*windmill *based on $T(V)$ is a k-hypertree whose hyperedges are the sets of vertices determined by the root-to-leaf paths of $T(V)$. $T(V)$ is called the* representing tree *of the windmill. A 1-windmill is called a* star.

A windmill farm *is a vertex-disjoint collection of windmills, and its* representing forest *is the union of the representing trees of the windmills. A 1-windmill farm is called a* sky.

Consider the edges in the representing forest as directed towards the roots. We say that each vertex points *to its parent.*

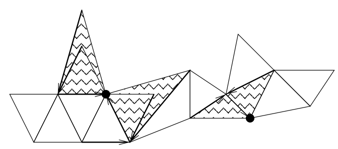

Figure 1: An example of a windmill farm inside a hypertree. The triangles are hyperedges. The directed edges correspond to the representing forest of the windmill farm, with the roots of the representing trees at the darkened vertices. The hyperedges that are in the windmill farm are shaded.

Definition 4. *The* windmill coverage *of a hypertree H with monotone weights w is the ratio of the maximum weight of any windmill farm in H to the weight of H:*

$$C(H, w) = \max_{M \subset H} \frac{w(M)}{w(H)} \text{ (where } M \text{ is a windmill farm).}$$

Intuitively, the windmill coverage measures "how well" a windmill farm can "approximate" a hypertree, by trying to capture as much of the weight of the hypertree as possible.

For an unweighted hypertree H, the windmill coverage is the greatest lower bound of the windmill coverage over all possible non-negative weights of H. Specifically, the windmill coverage of H is $C(H) = \min_w C(H, w)$.

We can also talk about the windmill coverage of a particular width k, which is the greatest lower bound of the windmill coverage over all k-hypertrees.

3 Problem 1: finding the windmill coverage of a weighted hypertree

We present a dynamic program to find the maximum weight windmill farm in a weighted hypertree. Note that finding the maximum weight windmill farm in arbitrary hypergraphs is NP-hard. However, in low treewidth graphs (equivalently, hypertrees), many NP-hard problems can be solved in polynomial time using dynamic programming [Bod93].

The key idea is that the hypertrees have separators[2] of size equal to the treewidth. The part of the solution to a problem that resides on one side of the separator can be found almost independently of the part of the solution on another side of the separator. The small dependence can be summarized in space proportional to some function $f(k)$, where k is the size of the separator, not of the entire hypertree. Thus, when k is a small constant, the dynamic program runs in time linearly proportional to the size of the hypertree.

3.1 Special case: treewidth 1

Before we give the general dynamic program of finding the maximum windmill farm for arbitrary hypertrees, let us consider the special case of treewidth 1: finding a maximum sky (1-windmill farm) in a tree (1-hypertree).

Suppose we are given a weighted tree (T, w), and we are trying to find the maximum weight sky in T. Assume that T is rooted arbitrarily, and let us try to construct a maximum sky in T top-down. Suppose that we have already constructed a partial sky S' for all of T except the subtree T_v rooted at some vertex v. What is the best way to finish constructing the sky in T_v to maximize the sky coverage given S'? Note that the answer does not depend on all of S', but only on how v is connected in S'. How v is connected in S' can be summarized (without losing information) into one of three states. The states are numbered 0, 1, and 2, so that higher states mean that v is less connected in S' and may connect more freely to vertices in T_v.

- **state 0**: v is connected to its parent, and its parent is connected to another vertex
- **state 1**: v is connected to its parent, but its parent is not connected to any other vertex
- **state 2**: v is disconnected from its parent

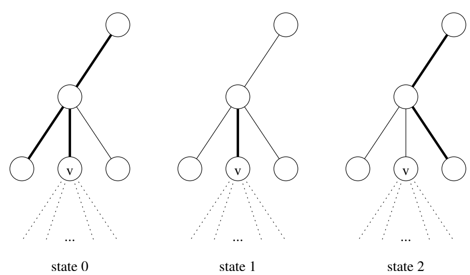

Figure 2: The possible states of a vertex v. Edges already in the partial sky S' are in bold. The dotted edges represent the subtree T_v rooted at v in which S' will be extended.

Now, we present a dynamic program to find the maximum sky in T. Let $f(v, q)$ denote the maximum weight that can be added to a partial sky that induces state $\leq q$ on vertex v. As q decreases, there are more restrictions on the extension. Then the sky coverage of T is $f(\mathrm{root}(T), 2)$. To compute $f(v, q)$, we imagine extending the star that contains v into the subtree T_v, and then recursively building the rest of the sky in T_v by invoking $f(\cdot)$ on the children of v, denoted $c_1, \ldots, c_n$. The following describes how to optimally extend the star at v to maximize $f(v, q)$ for each state q.

- **state 0**: Because v is already connected to its parent, and its parent is connected to another vertex, v may not be connected to any of its children. The extension of the sky is completed by invoking $f(c_i, 2)$ for all children. There is only one possible extension.
- **state 1**: Because v is connected to only its parent, we may join v with arbitrarily many children. If we connect c_i to v, we invoke $f(c_i, 0)$. If we don't connect c_i to v, then we invoke $f(c_i, 2)$. For each

[2] A set of vertices that disconnects a hypertree.

child c_i, we simply choose the option that contributes the most weight. Note that the null extension for state 0 is included as one of these possibilities.

- **state 2**: Certainly, we can use the extension for state 1. In addition, we can also connect a single child c_i to v, and then invoke $f(c_i, 1)$ for that child c_i and $f(c_j, 2)$ for $j \neq i$. Again, we choose the extension that maximizes the contributed weight.

The exact recurrences follow:

$$\begin{aligned} f(v,0) &= \sum_{i=1}^{n} f(c_i, 2) \\ f(v,1) &= \sum_{i=1}^{n} \max(f(c_i,2), w(v,c_i) + f(c_i,0)) \\ f(v,2) &= \max\big(f(v,1), f(v,0) + \max_{1\le i\le n} w(v,c_i) + f(c_i,1) - f(c_i,2)\big) \end{aligned}$$

The running time of this dynamic program is $O(|V|)$.

3.2 Windmill coverage in hypertrees

Now we turn our attention to the general case of finding the maximum windmill farm in a hypertree with non-negative weights. The basic idea is the same as for treewidth 1, but the states must be more expressive to capture the complexity of windmill farms in hypertrees. We shall construct a windmill farm in a given hypertree by constructing its representing forest, which is a directed forest over the vertices. Recall that the hyperedges of the windmill farm corresponding to this representing forest are the sets of vertices in the root-to-leaf paths in the forest.

Let (H, w) be a weighted hypertree and T_H its rooted tree structure over the hyperedges of H. In the future, we will be speaking of parent-child relationships in two contexts. Parent-child relationships between hyperedges refers to T_H, and parent-child relationships between vertices refers to the representing forest we are constructing. We seek a systematic procedure for constructing a representing forest, where the decisions made at each step of the procedure depend very little on the decisions made so far. At each invocation of this procedure we will add directed edges to the existing representing forest, so that each candidate directed edge is considered in exactly one invocation.

For every vertex v not in the root hyperedge h_{root}, there is some hyperedge h such that v is in $h - \mathrm{parent}(h)$. This follows by considering the Graham reduction of H bottom-up from the leaves of T_H. For every candidate directed edge $(u \to v)$ that can be added to the representing forest (excluding the case where $u, v \in h_{\mathrm{root}}$), there is some hyperedge h that contains both u and v, such that either u or v is in $h - \mathrm{parent}(h)$. In some sense, this hyperedge is "responsible" for $(u \to v)$. Our plan is to choose some initial representing forest in h_{root}, and then in turn consider each hyperedge h according to a top-down traversal of T_H, selectively adding the directed edges for which h is responsible.

Let h be a hyperedge and $c_1, \ldots, c_n$ be the children hyperedges of h. Suppose we have constructed a partial representing forest by considering all hyperedges excluding those in the sub-hypertrees rooted at $c_i, \ldots, c_n$. The optimal way to finish constructing the representing forest depends only on the state of h, detailed below. Given the state of h, we first extend the representing forest to some directed edges for which c_i are responsible. Then, we recursively invoke the procedure (twice) to add the directed edges for which the subtrees of c_i's children are responsible and the directed edges for which the subtrees of children $c_{i+1}, \ldots, c_n$ are responsible.

Of course, we must enforce that the directed edges form a valid representing forest:

- Ensure that all root-to-leaf paths stay within a single hyperedge. Then we guarantee a k-windmill farm. This is not too restrictive either, since the only root-to-leaf paths that can capture weight are those that are entirely in a single hyperedge.
- Ensure that no cycles in the representing forest are formed.

The state of an hyperedge h ought to be able to capture the relevant part of the partial representing forest. We summarize by remembering where each vertex in h points. Think of the root of a representing tree as pointing to itself. Each vertex v either points to some vertex in h, points to a vertex outside h (v is *blocked*), or does not point to any vertex yet (v is *free*).

The state of an hyperedge h is (R, F):

- R is the representing forest in h (set of directed edges that lie entirely in h)
- F is the set of free vertices

Any vertex that is not in F and does not point to any vertex is said to be *blocked*.

In the dynamic program, we compute $f(h, i, R, F)$, which is the maximum weight contribution by extending a representing forest that induces the state (R, F) on h to include directed edges for which the hyperedges in the subtrees $c_i, \ldots, c_n$ are responsible. Then, the weight of a maximum windmill farm in H is $\max_{(R,F)} f(h_{\text{root}}, 1, R, F)$, where we are taking the maximum over all initial representing forests in h_{root}.

Since H is a hypertree with only maximal k-hyperedges, each overlap between a hyperedge h and one of h's children is of size k. Thus, there is exactly one vertex α_i in $h - c_i$ and exactly one vertex β_i in $c_i - h$.

$$f(h, n+1, R, F) = \begin{cases} w(h) & \text{if } R \text{ is a directed path through all the vertices of } h \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$f(h, n, R, F)$ is the bottom case where we consider no new directed edges, but instead add weight of h if h is captured by the representing forest, which can be determined by looking at R.

$$f(h, i, R, F) = \max_{\text{extention of R into } c_i} f(h, i+1, R', F') + f(c_i, 1, R_c, F_c), \tag{2}$$

To extend R, we do the following two steps. Figure 3 shows an example of extending R.

1. Partition the free vertices F into three sets F_1, F_2, and F_3.
   - For $v \in F_1$, keep v free in (R', F'). v will be blocked in (R_c, F_c). It is important to enforce that F_1 includes the root of α_i (if that vertex exists in c_i), so that the root-to-leaf path to α_i will stay in a single hyperedge.
   - For $v \in F_2$, keep v free in (R_c, F_c). In this case, v will not point to any vertex in c_i or h, but rather some descendant of c_i. v will be blocked in (R', F') so the invocation to $f(h, i+1, R, F)$ won't try to point v to any vertex not in the hyperedges of the subtree rooted at c_i.
   - For $v \in F_3$, point v to β_i. v will be blocked in (R', F') so the invocation to $f(h, i+1, R, F)$ won't try to point v to any vertex (v already points to β_i).
2. For β_i, choose one of the following options:
   - Point β_i to a vertex v_b in c_i such that the root of v_b is in c_i, so that this root-to-leaf path stays within c_i. Also, make sure that $(\beta_i \to v_b)$ does not create a cycle.

- Keep β_i free in (R_c, F_c), so that β_i will point to some vertex in a descendant of c_i.

The states (R', F') and (R_c, F_c) are computed as follows:

- $R' = R$.
- R_c includes R and any edges connecting β_i, but excluding any edges connecting α_i.
- $F' = F_1$.
- F_c includes F_2 and also β_i if β_i is free in (R_c, F_c).

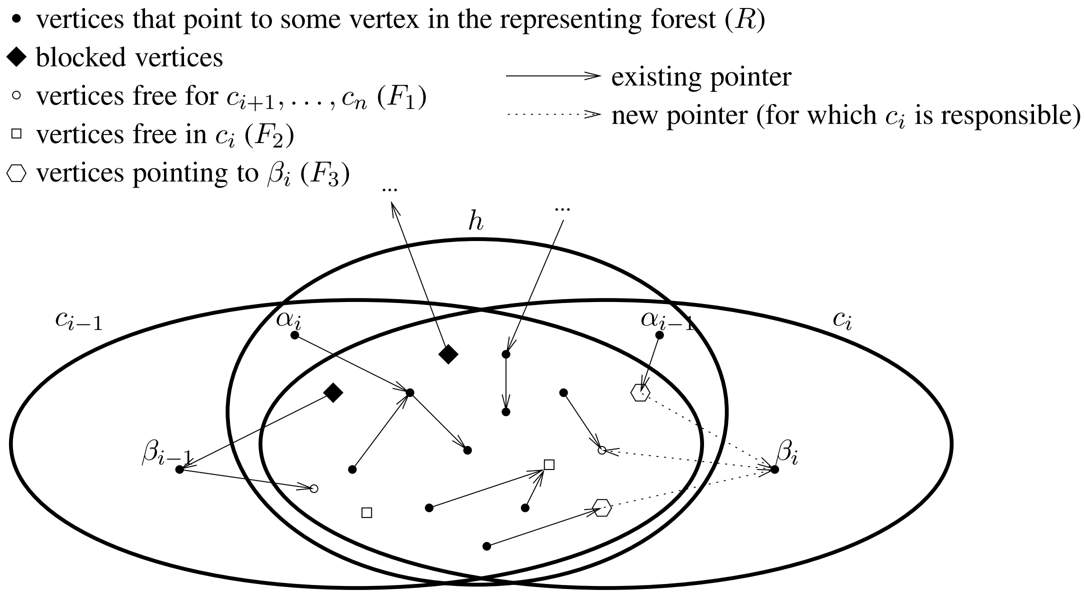

Figure 3: An example of a call to $f(h, i, R, F)$.

The running time of this dynamic program is $O((k+3)^{k+1}|V|)$.

4 Problem 2: finding the windmill coverage of an unweighted hypertree

We now describe a linear program based on the dynamic program used to solve Problem 1. The dynamic program involves computing many values (in the maximum sky dynamic program, $f(v, q)$ for all vertices v and $q \in \{0, 1, 2\}$). These are the dynamic programming (DP) states. The value of each DP state is equal to an expression involving only addition and $\max$ operations of the values of other DP states.

The general technique for converting a dynamic program of this form into a linear program (LP) is to make each DP state a LP variable. Values known in the DP but unknown in the LP (such as hyperedge weights) also become LP variables. Each expression for computing the value of a DP state is translated into several LP inequalities and perhaps more LP variables.

An expression of the form $A = B + C + D$ translates into the same LP equation. The expression $A = \max(B, C, D)$ translates into three LP inequalities: $A \geq B$, $A \geq C$, and $A \geq D$. A more complex expression $A = \max(B, C, D) + E$ is equivalent to the translation of two expressions $A = N + E$ and $N = \max(B, C, D)$, where N is a fresh variable.

Following the technique described above, let us construct the LP. The variables of the LP are the DP states $f(v, q)$, the weights of the edges $w(v_1, v_2)$, and a new variable for each $\max$ expression. For example, the DP equation for $f(v, 1)$ yields the following LP inequalities, where z_i's are fresh variables:

$$
\begin{aligned}
f(v,1) &= \sum_{i=1}^{n} z_i \\
\forall i, z_i &\geq f(c_i,2) \\
\forall i, z_i &\geq w(v,c_i) + f(c_i,0)
\end{aligned}
$$

In a feasible settings of the variables, $f(v,q)$ is a possible weight for a sky in the subtree T_v with the connectedness of v specifi ed by q.

Finally, we add one additional equation requiring that all edge weights sum to 1. Because we are trying to minimize the ratio of the maximum weight sky to the weight of the tree, we can ignore the division by the weight of the tree in computing the sky coverage. The objective function is to minimize the single variable representing the weight of the maximum sky, $f(v, \mathrm{root}(T))$.

Constructing a linear program to fi nd the windmill coverage of an unweighted hypertree H is the procedure for converting the dynamic program into a linear program is the same as the one discussed for the sky coverage case.

4.1 Necessity of a linear program

To solve Problem 2, we cannot devise a dynamic program in which subproblems would involve solving small linear programs instead of solving one giant linear program with the same size as the dynamic program. For example, in the sky coverage case, we would like to express $f(v,q)$ (the sky coverage of the tree structure rooted at v) in terms of the sky coverage of subtree structures, and minimize $f(v,q)$ by solving an LP local to v.

The problem is that $f(v,q)$ for different values of q are not independent from each other. Their values correspond to different setting of weights, and thus cannot be achieved at the same time. An expression therefore cannot blindly reference $f(v,0)$ and $f(v,1)$ independently to concoct a set of new weights, because the weights corresponding to $f(v,0)$ and the weights corresponding to $f(v,1)$ might differ.

Each assignment of weights in T_v can be summarized as a triple $(f(v,0), f(v,1), f(v,2))$. Each triple can be seen as a feasible three-dimensional point. We only care about storing for v the best points. The most compact representation is to store, for two of the coordinates, the largest value of the third coordinate. Notice that if the feasible points were one-dimensional, we would only need to store a single value. But for three-dimensional points in our case, we need to store a value for each of many points in the plane, which is prohibitively expensive.

5 Problem 3: fi nding the windmill coverage

5.1 Special case: treewidth 1

Now, we attempt to fi nd the sky coverage (the sky coverage of the absolute “worst” weighted tree). The sky coverage of a tree T is at most the sky coverage of a subtree $T' \subset T$, since the worst assignment of weights for T' can be augmented to T by simply assigning zero weight to edges not in T'. Therefore, let us consider complete trees of breadth b and depth d, denoted $T(b,d)$. If we let b and d tend to infi nity, then the sky coverage of $T(b,d)$ tends to the sky coverage. Table 1 in the appendix shows some results obtained for various values of b and d.

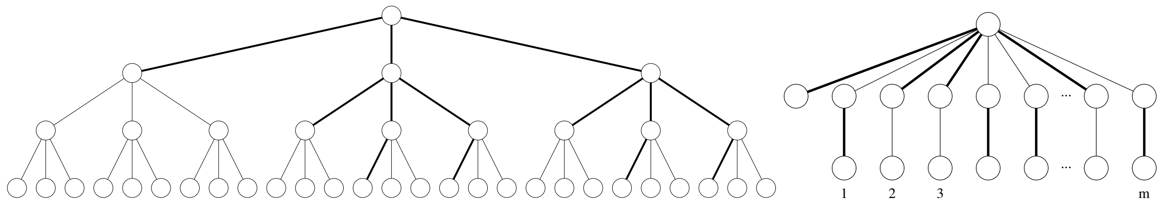

Figure 4: (a) An example of a worst assignment of weights for $T(3,3)$. Bold edges have weight 1. Non-bold edges have weight 0. (b) Consider this sequence of weighted trees T_m, where all weights are equal. An example of a maximum sky in T_m is marked by bold edges. The sky coverage of T_m is equal to $\frac{m+1}{2m-1}$, which approaches the lower bound of $1/2$ as $m \to \infty$.

5.1.1 Properties of skies and trees

Though we have achieved our goal for finding a worst weighted tree that minimizes the sky coverage, we would like to still study the assignment of weights to some *given* tree structure. We found empirically that the worst assignment of weights were always binary (all edge weights are 0 or 1) and that the subforest induced by the weight 1 edges forms a tree. Furthermore, all the induced subtrees satisfy the surprising property that each non-leaf vertex in the subtree is connected to exactly one leaf vertex. See Figure 4 for an example. We conjecture that this is true for all tree structures, and have proved that it is true for certain class of tree structures, including trees with the property.

5.2 General case

5.2.1 Complete structure

As in the sky coverage case (Section 5.1), we run the linear program over complete structures controlled by some breadth b and depth d. Define $H_c(k,b,d)$ to be the family of *complete k-hypertree structures*. $H_c(k,b,d)$ is the k-hypertree corresponding to the tree structure T_H constructed in the following way. T_H has has depth d. The root has $(k+1)b$ children (b on each separator) and all other hyperedges excluding the leaves have kb children (b on each separator). Since every k-hypertree is contained in some $H_c(k,b,d)$, the windmill coverage is obtained as b and d approach infinity. Table 2 in the appendix summarizes the results obtained by the program for treewidth $k=2$.

We can prune the hypertree without changing the windmill coverage. Let L be the set of leaves in the tree structure of $H_c(k,b,d)$ that all share the same separator s with the rest of the hypertree. We claim that there is no need for L to contain more than one hyperedge, since any maximum windmill farm will either contain all of the hyperedges in L or none of them. If L contained more than one hyperedge, then we could achieve the same windmill coverage bound with one hyperedge in L.

The proof of the claim goes as follows. Suppose a maximum windmill farm M contains $h \in L$. Let $v \in h$ be the single vertex not in the separator s. Let R be the representing forest of M projected onto h. R must be a directed path that goes through all the vertices in h. This path might as well end in v, or else there is no chance of capturing any hyperedges adjacent to h. Then M can contain all hyperedges in L simply by extending the path in the representing forest through s to each of the vertex leaves in the other hyperedges of L.

Unlike in the sky coverage case, the worst assignment of weights obtained by the linear program for $H_c(k,b,d)$ are not binary in general. In fact, it is not always possible to achieve the windmill coverage bound of a hypertree H using only binary weights. For instance, the windmill coverage of $H_c(2,1,2)$ is

$4/11$, but the worst windmill coverage obtained using binary weights is $4/10$.

5.2.2 Exponentially decreasing weights with leaf restrictions

Using the results from these runs, we noticed two properties that the worst assignment of weights exhibits. First, non-leaf hyperedges in the tree structure generally are adjacent to a leaf hyeredge. The importance of leaves in creating the worst assignment of weights was also noted in Section 5.1.1. Second, the weights of the hyperedges decrease roughly exponentially as the depth increases linearly.

Based on these observations, we concocted a sequence of *weighted* hypertrees, which we call $H_e(k, b, d)$, which tightens the gap between lower and upper bounds of the windmill coverage. Note that unlike H_c, H_e is weighted. Figure 5 gives an example of $H_e(2, 2, 3)$.

The weights are halved with increasing depth, and each non-leaf hyperedge in the tree structure is adjacent to exactly one leaf hyperedge per separator. Formally,

- $H_e(k, b, 0)$ is the empty hypergraph.
- $H_e(k, b, d)$ includes a hyperedge h with weight 2^{d-1}, and for each of h's $k + 1$ separators, a leaf hyperedge with weight 2^{d-1} and $b - 1$ copies of $H_e(k, b, d - 1)$ (except that the number of separators for these subtrees is k instead of $k + 1$).

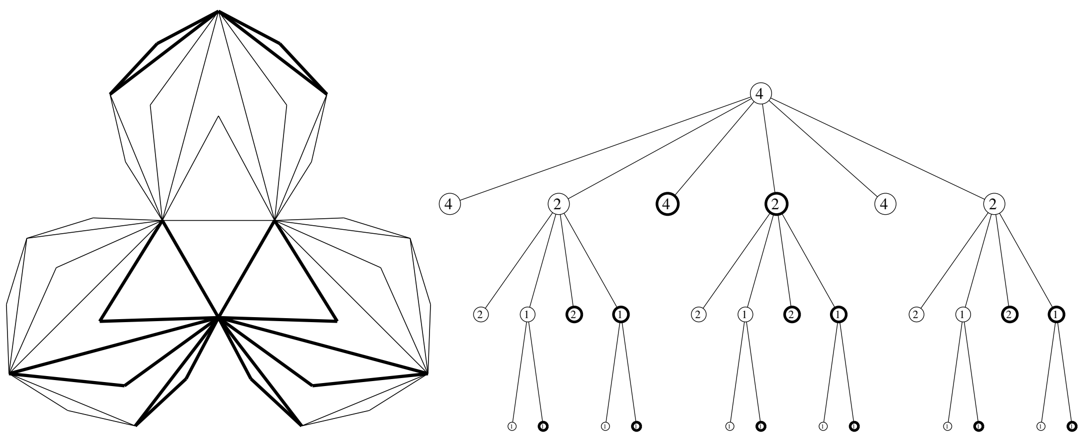

Figure 5: Two representations of $H_e(2, 2, 3)$. In (a), each triangle is an hyperedge, and the bold triangles represent the hyperedges in a maximum windmill farm. In (b), each hyperedge is a node in the tree structure, labelled with the weight of that hyperedge. The children of a hyperedge are grouped by separator, depicted by alternating bold and normal styles.

To find the windmill coverage of H_e, we used a specialized version of the dynamic program presented in Section 3.2, taking advantage of the regular structure. The specialized dynamic program runs in time linearly proportional to the depth d of the hypertree. We ran the program for successively increasing depths until the windmill coverage converged. It is interesting to note that convergence happens faster for larger width.

For $k > 1$, the minimum windmill coverage was obtained with $b = 2$. (For $k = 1$, $b = 3$ yielded a smaller windmill coverage.) The weights of a maximum windmill farm when $b = 2$ and $d \to \infty$:

6 Discussion

We tackle the problem of finding a "worst case" hypertree, a weighted hypertree that comes as close to the overall windmill coverage bound as possible, i.e., such that no windmill farm can capture much of the weight

k	lower bound $(1/(k+1)!)$	$\lim_{d\to\infty} C(H_e(k,2,d))$	previous upper bound $(1/(k+1))$
2	0.166667	0.2222222 = 2/9	0.33333
3	0.041667	0.0953932...	0.25
4	0.008333	0.0515625 = 33/640	0.2
5	0.001389	0.0258048 = 2016/78125	0.16667
6	0.000198	0.0123157...	0.14286

of the hypertree. Given weights on a hypertree (Problem 1), we can readily find the windmill coverage of that hypertree via dynamic programming. The challenge is finding the worst case assignment of weights (Problem 2). Here, we presented an novel approach to this problem by encoding the input weights of the dynamic program as variables in the linear program.

The hardest leap is Problem 3, where we must find not only the worst weights but the worst unweighted hypertree, which we do by considering increasingly larger complete hypertrees. By looking at the resulting worst case weights for particular hypertrees, we reap the intuitive properties that a worst case hypertree might have. Since the size of linear program grows exponentially with the depth of the hypertree, we use these properties to narrowed our search space. Eventually, we came up with a particular family of weighted hypertrees (H_e) that was much closer to the lower bound than our previous upper bound. Yet, the worst case windmill coverage ratio remains unresolved. We hope that further investigation of structures revealed here can help not only in finding "bad" examples, but also in strengthening the positive results and proving a higher lower bound on the windmill coverage.

Acknowledgments We are thankful to Erik Demaine for comments and Swastik Kopparty for helping with the analysis of the binary weights property in trees.

References

[Bod93] Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1193.

[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.

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

Appendix

$b\backslash d$	2	3	4	5	6	7
2	0.6667	0.6000	0.5556	0.5384	0.5294	0.5238
3	0.6000	0.5385	0.5172	0.5082	0.5040	0.5020
4	0.5714	0.5200	0.5063	0.5021	0.5007	0.5002

Table 1: Sky coverage for complete rooted tree structures $T(b, d)$.

$b\backslash d$	2	3	4	5	6	7
1	4/11=0.364	1/3=0.333	4/13=0.308	5/17=0.294	12/43=0.279	24/89=0.270
2	4/13=0.308	7/26=0.269	15/62=0.242	51/226=0.226		
3	7/23=0.304	10/38=0.263	18/78=0.231			
4	10/33=0.303	13/50=0.260	15/67=0.224			
5	13/43=0.302	16/62=0.258				
6	16/53=0.302					
7	19/63=0.302					
8	22/73=0.301					

Table 2: Windmill coverage coverage for complete tree structures $H_c(2, b, d)$.