Tree Checking for Sparse Complexes

Massimo Caboara[1], Sara Faridi[2,⋆], and Peter Selinger[3,⋆]

[1] University of Pisa, Italy, caboara@dm.unipi.it.
[2] Dalhousie University, Halifax, Canada, faridi@mathstat.dal.ca.
[3] Dalhousie University, Halifax, Canada, selinger@mathstat.dal.ca.

Abstract. We detail here the sparse variant of the algorithm sketched in [2] for checking if a simplicial complex is a tree. A full worst case complexity analysis is given and several optimizations are discussed. The practical complexity is discussed for some examples.

1 Introduction

The main goal of this paper is to give a detailed description and worst time complexity for a sparse variant of the tree-checking algorithm introduced in the paper [2]. For all the proofs in Sects. 2-3 we refer to [2].

Facet ideals were introduced in [4] as a method to study square-free monomial ideals, generalizing results in [9] and [8] on edge ideals of graphs. The idea is to associate a simplicial complex to a square-free monomial ideal, where each facet (maximal face) of the complex is the collection of variables that appear in a monomial in the minimal generating set of the ideal. The definition of a simplicial tree is a generalization of the concept of a graph-tree. Monomial ideals associated to trees have many properties that make them useful from an algebraic point of view [4, 7].

In Sect. 2 we briefly recall the notation and results of [2]. In Sect. 3 we borrow from [2] a first version of the tree checking algorithm and we discuss its complexity. In Sect. 4 we present the main result of this paper, the full description of a variant of the tree checking algorithm optimized for sparse complexes and its worst case complexity. This variant has been briefly sketched in [2]. A brief subsection proposing further developments ends the paper.

Implementations. The algorithms described in this paper have first been coded in CoCoAL, the CoCoA system programming language (http://cocoa.dima.unige.it/). These prototypical implementations can be downloaded from [1]. Much more efficient (but less user friendly) C++ implementations have been developed for several versions of Algorithm 3.1 using the CoCoALib framework (http://cocoa.dima.unige.it/cocoalib/). The C++ code is also available at the website [1]. The code will be available in the CoCoA system from version 4.6 onwards. The code for the sparse variant is still a rough prototype, but already useful to test some example.

⋆ Research supported by NSERC

2 Simplicial Complexes and Trees

We define the basic notions related to facet ideals. More details and examples can be found in [4, 5].

Definition 2.1 (Simplicial complex, facet). A *simplicial complex* Δ over a finite set of vertices V is a collection of subsets of V, with the property that if $F \in \Delta$ then all subsets of F are also in Δ. An element of Δ is called a *face* of Δ, and the maximal faces are called *facets* of Δ.

Since we are usually only interested in the facets, rather than all faces, of a simplicial complex, it will be convenient to work with the following definition:

Definition 2.2 (Facet complex). A *facet complex* over a finite set of vertices V is a set Δ of subsets of V, such that for all $F, G \in \Delta$, $F \subseteq G$ implies $F = G$. Each $F \in \Delta$ is called a *facet* of Δ.

The set of facets of a simplicial complex forms a facet complex. Conversely, the set of subsets of the facets of a facet complex is a simplicial complex. This defines a one-to-one correspondence between simplicial complexes and facet complexes. In this paper, we will work primarily with facet complexes.

Let k be a field. To a facet complex over a vertex set $\{v_1, \dots, v_n\}$, one can uniquely associate an ideal $\mathcal{F}(\Delta)$ in the polynomial ring $k[x_1, \dots, x_n]$, where $\mathcal{F}(\Delta)$ is generated by all the square-free monomials $x_{i_1} \dots x_{i_s}$, with $\{v_{i_1}, \dots, v_{i_s}\}$ a facet of Δ. This ideal is called the *facet ideal* of Δ.

From now on, we will often ease the notation by denoting facets of a complex by their corresponding monomials; for example, we write xyz for the facet $\{x, y, z\}$.

We now generalize some notions from graph theory to facet complexes. Note that a graph can be regarded as a special kind of facet complex, namely one in which each facet has cardinality 2.

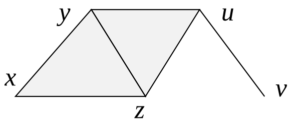
Definition 2.3 (Path, connected facet complex). Let Δ be a facet complex. A sequence of facets $F_1, \dots, F_n$ is called a *path* if for all $i = 1, \dots, n-1$, $F_i \cap F_{i+1} \neq \emptyset$. We say that two facets F and G are *connected* in Δ if there exists a path $F_1, \dots, F_n$ with $F_1 = F$ and $F_n = G$. Finally, we say that Δ is *connected* if every pair of facets is connected.

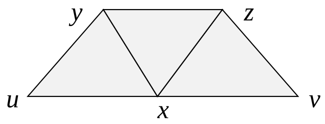
Notation 2.1. If F, G and H are facets of Δ, $H \leqslant_F G$ means that $H \cap F \subseteq G \cap F$. The relation $\leqslant_F$ defines a preorder (reflexive and transitive relation) on the facet set of Δ.

Definition 2.4 (Leaf, joint). Let F be a facet of a facet complex Δ. Then F is called a *leaf* of Δ if either F is the only facet of Δ, or else there exists some $G \in \Delta \setminus \{F\}$ such that for all $H \in \Delta \setminus \{F\}$, we have $H \leqslant_F G$.

It follows immediately from the definition that every leaf F contains at least one *free vertex*, i.e., a vertex that belongs to no other facet.

Example 2.1. In the facet complex $\Delta = \{xyz, yzu, uv\}$, xyz and uv are leaves, but yzu is not a leaf. Similarly, in $\Delta' = \{xyu, xyz, xzv\}$, the only leaves are xyu and xzv.

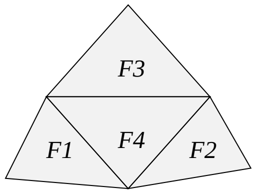
$\Delta =$ 

$\Delta' =$ 

Definition 2.5 (Forest, tree). A facet complex Δ is a *forest* if every nonempty subset of Δ has a leaf. A connected forest is called a *tree* (or sometimes a *simplicial tree* to distinguish it from a tree in the graph-theoretic sense).

It is clear that any facet complex of cardinality one or two is a forest. When Δ is a graph, the notion of a simplicial tree coincides with that of a graph-theoretic tree.

Example 2.2. The facet complexes in Example 2.1 are trees. The facet complex Δ'' pictured below has three leaves F_1, F_2 and F_3; however, it is not a tree, because if one removes the facet F_4, the remaining facet complex has no leaf.

$\Delta'' =$ 

Alternatively, one could define a cycle and define a tree to be a connected complex that contains no cycles.

Definition 2.6 (Cycle). A *cycle* is a nonempty facet complex that has no leaf, but every proper subset of it has a leaf.

For example, the subcomplex $\{F_1, F_2, F_3\}$ is a cycle in Example 2.2, but the whole complex is not.

It follows immediately that a facet complex is a forest if and only if it contains no cycles.

2.1 Characterization of Trees

We now consider the problem of deciding whether or not a given facet complex is a tree.

Note that the naïve algorithm (namely, checking whether every non-empty subset has a leaf) is extremely inefficient: for a facet complex of n facets, there are $2^n - 1$ subsets to check. Also note that the definition of a tree is not inductive in any obvious way: for instance, attaching a single leaf to a tree need not yield a tree, as Example 2.2 shows. This seems to rule out an easy recursive algorithm.

Nevertheless, we demonstrate that the decision problem for simplicial trees can be solved efficiently. This is done via a characterization of trees given in this section.

Definition 2.7 (Paths and connectedness outside V). Let Δ be a facet complex, and let V be a set of vertices. We say that a sequence of facets $H_1, \dots, H_n \in \Delta$ is a *path outside* V in Δ if for all $i = 1, \dots, n-1$, $(H_i \cap H_{i+1}) \setminus V \neq \emptyset$. We say that two facets $F, G \in \Delta$ are *connected outside* V in Δ if there exists a path $H_1, \dots, H_n$ outside V in Δ such that $H_1 = F$ and $H_n = G$.

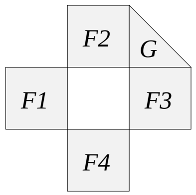
Note that in case $V = \emptyset$, this coincides with the definition of connectedness from Definition 2.3.

Notation 2.2. If F, G_1, G_2 are three distinct facets of Δ, then we define $\Delta_F^{G_1,G_2}$ to be the following subset of Δ:

$$\Delta_F^{G_1,G_2} = \{H \in \Delta \mid H \cap F = G_1 \cap G_2\} \cup \{G_1, G_2\}.$$

Definition 2.8 (Triple condition). Let Δ be a facet complex. We say a triple of facets $\langle F, G_1, G_2 \rangle$ satisfies the *triple condition* if $G_1 \not\leq_F G_2$ and $G_2 \not\leq_F G_1$, and if G_1 and G_2 are connected outside F in the facet complex $\Delta_F^{G_1,G_2}$.

Example 2.3. Consider the facet complex

$\Delta =$ 

The triple $\langle F_1, F_2, F_4 \rangle$ satisfies the triple condition. This is because $F_4 \not\leq_{F_1} F_2$ and $F_2 \not\leq_{F_1} F_4$. Moreover $\Delta_{F_1}^{F_2,F_4} = \{F_2, F_3, F_4, G\}$, and a path connecting F_2 and F_4 outside F_1 is F_2, F_3, F_4.

However, $\langle G, F_2, F_3 \rangle$ does not satisfy the triple condition, since $\Delta_G^{F_2,F_3} = \{F_2, F_3\}$, and F_2 and F_3 are not connected outside G.

Let Δ be a connected facet complex. Then the triple condition determines whether or not a triple of facets belongs to a cycle contained in Δ ([2] Proposition 4.5). In particular we have the following ([2] Theorem 4.6) criterion that determines if a given facet complex is a tree.

Theorem 2.3 (Main Theorem). *A connected facet complex Δ is a tree if and only if no triple of facets in Δ satisfies the triple condition.*

3 A Polynomial-time Tree Decision Algorithm

By Theorem 2.3, to check if a facet complex $\Delta = \{G_1, \ldots, G_l\}$ is a tree, we only need to check the triple condition for all triples of elements of Δ. The checks themselves are straightforward. Since the triple condition for $\langle F, G, G' \rangle$ is clearly unchanged if one switches G and G', we can limit triple checking to the elements of the set $\{\langle F, G_i, G_j \rangle \in \Delta^3 \mid G_i \neq F \neq G_j, i < j\}$. The procedures for the basic steps follow immediately from the earlier definitions.

Algorithm 3.1 (Standard algorithm).
Input: a connected facet complex $\Delta = \{G_1, \ldots, G_l\}$ with n vertices.
Output: **True** if Δ is a tree, **False** otherwise.

1. For each triple $\langle F, G, G' \rangle \in \{\langle F, G_i, G_j \rangle \in \Delta^3 \mid G_i \neq F \neq G_j, i < j\}$
   (a) If $G \leqslant_F G'$ or $G' \leqslant_F G$, continue with the next triple.
   (b) Build $\Delta_F^{G,G'}$.
   (c) If G and G' are connected outside F in $\Delta_F^{G,G'}$, return **False**.
2. Return **True**.

The correctness of this algorithm is an immediate consequence of Theorem 2.3. The algorithm uses very little memory; the input Δ requires nl bits, and $\Delta_F^{G,G'} \subseteq \Delta$ requires l bits. The memory required to perform the connectedness check and to store the various counters is negligible. Thus, memory locality is good, and the computations can generally take place in the cache.

Remark 3.1. In the process of checking the triple condition for a triple $\langle F, G, G' \rangle$ that is part of a cycle, we build a connection path outside F. Clearly, any such path can be reduced to a *minimal* connected path $\{H_1, \ldots, H_n\}$ outside F for G, G', and $\{F, H_1, \ldots, H_n\}$ forms a cycle. Therefore, an easy modification of Algorithm 3.1 allow us to produce the set of all the facets $F \in \Delta$ that are part of some cycle, and a cycle $\Delta'_F \supseteq \{F\}$ for each of them.

3.1 Complexity

For each triple it is trivial to see that steps (a) and (b) can be performed with cost $O(n)$ and $O(nl)$ respectively. For step (c), the following holds.

Lemma 3.1 (Relation algorithm). *Let Δ be a facet complex with l facets over n variables such that F, G, G' are distinct facets of Δ. The connectedness outside F of $G, G' \in \Delta$ can be determined with time cost $O(nl)$.*

Proof. First of all we substitute Δ with the set $\{H - F \mid H \in \Delta\}$. We then define $n+1$ equivalence relations $P_0, \ldots, P_n$ on the set $\{1, \ldots, l\}$. P_0 is the identity relation, i.e., each equivalence class is a singleton. For each $j = 1, \ldots, n$, consider the vertex v_j and the set $X_j = \{i \mid v_j \in F_i\}$. Let P_j be the smallest equivalence relation such that $P_{j-1} \subseteq P_j$ and such that for all $i, i' \in X_j$, $(i, i') \in P_j$. Then facets F_i and $F_{i'}$ are connected if and only if $(i, i') \in P_n$. With a suitable data structure for representing equivalence relations (e.g., the relation associated to the partition $\{1\}$, $\{2,3\}$,$\{4,5,6\}$ of $\{1, \ldots, 6\}$ can be represented by the array of integers $[1,2,2,4,4,4]$), the complexity of the procedure above is $O(nl)$.

Consequently, step (c) of the tree decision algorithm can be performed at cost $O(nl)$. Thus, the total complexity of the tree decision algorithm is as follows: in the worst case we have to check $3 \cdot \binom{l}{3} = \frac{l(l-1)(l-2)}{2} = O(l^3)$ triples. The complexity of the steps (a)–(c) is $O(nl)$ and hence the total complexity of the algorithm is $O(nl^4)$.

Example 3.1. Consider the facet complex $\Delta = \{xy, xz, yz, yu, zt\}$. We have to check $3 \cdot \binom{5}{3} = 30$ triples. We start with the triple $\langle xy, xz, yz \rangle$.

- $xz \nleqslant_{xy} yz$ since $xy \cap xz = x \nsubseteq y = xy \cap yz$. Similarly $yz \nleqslant_{xy} xz$.
- xz and yz are connected outside xy in the complex $\Delta_{xy}^{xz,yz} = \{zt, xz, yz\}$.

We have hence discovered that Δ is not a tree. An unlucky choice of facets could have brought about the checking of 27 useless triples before this discovery, the other two useful triples being $\langle yz, xy, xz \rangle$ and $\langle xz, xy, yz \rangle$.

Remark 3.2. The relation algorithm for checking connectedness has very good worst case complexity. It is not so efficient in the average case, as shown below, see Table 2. Let us detail another algorithm for checking the connectedness of $F, G \in \Delta' = \{H_1, \ldots, H_l\}$: we examine $H_1, H_2, \ldots$ the elements of $\Delta - \{F, G\}$ and pose $F' = F$; if $H_i \cap F' \neq \emptyset$ we substitute F' with $F' \cup H_i$ and delete H_i from Δ. We repeat the previous procedure until $F' \cap G \neq \emptyset$ (F and G are connected) or $\Delta = \emptyset$ (F and G are not connected). Worst case complexity of this *list* algorithm is $O(nl^2)$, but we will see that it seems to be quite efficient in the average case.

Some Statistics

The facet complex $\{x_i x_{i+1} x_{i+2} \cdots x_{i+n} \mid i = 1, \ldots, m\}$ is trivially a tree. We call it the *line* n/m complex. The facet complex

$$\{yx_{11}x_{12}x_{13}, \ldots, yx_{1n-2}x_{1n-1}x_{1n}, \ldots, yx_{m1}x_{m2}x_{m3}, \ldots, yx_{mn-2}x_{mn-1}x_{mn}\}$$

is also trivially a tree, and we call it *star line* m/n. A random facet complex over $m+10$ vertices, with m facets, each of which contains from n to k vertices, will be called *random* $m/n/k$ complex. It is extremely unlikely for such a complex to be a tree if $m > 30$ and $n, k > 5$.

Let us examine the connectedness check (Conn. Chk.) timings for the list and relation algorithms, compared to total timings, for some examples:

The list algorithm looks better in the average (sparse or almost sparse) case than the relation algorithm with respect to practical complexity. In the following examples, we will check connectedness with the list algorithm.

The following table gives the statistics for the checking of *every* triple for some random n/k examples. The Conn. Chk. and Triple. Cond. columns give the percentage of triples (against the total number of triples in both cases) that need a connectedness check or satisfy the triple condition respectively. The Tot. Time and Conn. Chk. Time columns give the total time spent and the time spent checking connectedness.

Table 1. List and Relation algorithms - Conn. Chk. Timings

Example	List algorithm Time Total/Conn. Chk.	Relation algorithm Time Total/Conn. Chk.
Rand.100/5/10	2.9s/0.5s	26.0s/23.6s
Rand. 200/5/10	21.2s/4.4s	467s/448s
Line 400/3	14.2s/1.1s	30.0s/16.3s
Line 400/40	115.1s/31.2s	2,418s/2,332s

Table 2. Random Examples

Example	Conn. Chk.	Triple. Cond.	Tot. Time	Conn. Chk. Time
Rand. 100/5/10	14%	14%	2.9s	0.6s
Rand. 100/20/40	100%	98%	9.4s	1.3s
Rand. 200/5/10	6%	6%	21.0s	4.4s
Rand. 200/20/40	94%	93%	132s	9.7s
Rand. 200/120/120	100%	100%	138s	10.2s
Rand. 400/5/20	7%	7%	625s	106s
Rand. 400/40/80	99%	99%	2,637s	90s

The more "dense" a random complex is, the higher the percentage of triples for which connectedness has to be checked and that satisfy the triple condition. It is exceedingly difficult for a random complex to be a tree, and the detection of a triple satisfying the triple condition is usually quite easy.

Tree examples are the hard cases, since every triple has to be checked.

Table 3. Tree Examples

Example	Conn. Chk.	Time
line 400/3	0.005%	14.2s
line 400/40	2%	115s
star line 4/100	0.01%	12.7s
star line 10/100	0.0003%	300s

3.2 Optimization

The runtime of Algorithm 3.1 can be improved by introducing some optimizations. First, note that if F is a facet such that no triple $\langle F, G, G' \rangle$ satisfies the triple condition, then F cannot be part of any cycle of Δ. Therefore, F can be removed from Δ, reducing the number of subsequent triple checks. We refer to this optimization as the *removal of useless facets.*

Remark 3.3. The facet order in a complex can be crucial when using the useless facet optimization, as shown in Table 4 below.

An important special case of a "useless facet" is a reducible leaf, as captured in the following definition:

Definition 3.1 (Reducible leaf). A facet F of a facet complex Δ is called a *reducible leaf* if for all $G, G' \in \Delta$, either $G \leqslant_F G'$ or $G' \leqslant_F G$.

A reducible leaf is called a "good leaf" by Zheng [10].

Remark 3.4. The facet F is a reducible leaf of Δ if and only if F is a leaf of every $\Delta' \subseteq \Delta$ with $F \in \Delta'$.

The remark immediately implies that a reducible leaf cannot be part of a cycle. Thus, it can be removed from Δ, and the algorithm can then be recursively applied to $\Delta \setminus \{F\}$. We were not able to find a tree without a reducible leaf; in fact, Zheng [10] conjectured that this is always the case. Checking whether a given facet F is a reducible leaf requires ordering all facets with respect to $\leqslant_F$, which takes $O(nl \log l)$ steps. A reducible leaf can thus be found in time $O\big(nl^2 \log l\big)$. Therefore, if Zheng's conjecture is true, the tree problem can be decided in time $O\big(nl^3 \log l\big)$. But even if the conjecture is not true, removing all reducible leaves at the beginning of Algorithm 3.1 is still a worthwhile optimization.

4 Optimization for Sparse Complexes

Let Δ be a facet complex with l facets. If every $F \in \Delta$ intersects a substantial ($\approx l$) number of facets, then the number of triples that satisfy the triple condition is probably high and our algorithm is usually able to detect one of them easily. If this does not happen, we can exploit the facet complex "sparseness" in our algorithm. For the remainder of this subsection, Δ will be a facet complex with l facets over n vertices.

4.1 Sparse Algorithm

Precomputing the incidence matrix for the graph describing the connectedness relation for the complex Δ allows us to use very efficient versions of all the sub procedures. The implementation of the sparse variant of the algorithm is still not complete, but we give here a full description, a complexity analysis and a prototype implementation, plus some examples.

Notation 4.1. Let Δ be a facet complex and $F \in \Delta$. We denote by d_F the cardinality of the set $\{H \in \Delta \mid H \cap F \neq \emptyset\}$. We denote by v_F the number of vertices in F.

Note that for $F \in \Delta$ we always have $d_F \leqslant l$ and $v_F \leqslant n$. Note also that the with a suitable implementation, costs for the intersection and equality/inequality operations for F, G are $O(\min\{v_F, v_G\})$.

Definition 4.1 (Connection block). Let $\Delta = \{F_1, \dots, F_l\}$ be a facet complex. The *connection block of* Δ, (CB_Δ), is the list of pairs $\langle i, \{j_1, \dots, j_{d_{F_i}}\}\rangle$, $1 \leqslant i \leqslant l$ where $\{F_{j_1}, \dots, F_{j_{d_{F_i}}}\}$ is the list of the facets connected to F_i. We call $\{j_1, \dots, j_{d_{F_i}}\}$ in $\langle i, \{j_1, \dots, j_{d_{F_i}}\}\rangle$ the i-th row of CB_Δ.

Note that CB_Δ is the incidence matrix for the graph describing the connectedness relation for the complex Δ. We denote the sum of the cardinality of the i-th rows of CB_Δ by E. Note that $E = \sum_{i=1}^{l} d_{F_i} = 2 \cdot \#(\text{edges in the graph})$. The space required to store CB_Δ is $O(E)$.

Notation 4.2. Let Δ be a facet complex and $F \in \Delta$. We denote by CB_Δ^F the connection block of the facet complex Δ when the connectedness relation is replaced by the connectedness outside F relation.

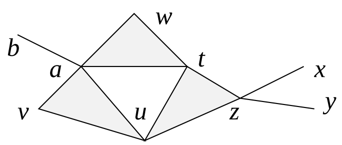
The space required to store CB_Δ^F is less or equal than the space necessary to store CB_Δ.

If we have $\Delta, \mathrm{CB}_\Delta$,and $\Delta' \subset \Delta$ and we want to build $\mathrm{CB}_{\Delta'}$ we can do that efficiently by marking in Δ all the elements in $\Delta - \Delta'$. When using $\mathrm{CB}_{\Delta'}$ to check connectedness, we work in CB_Δ but we only consider indices whose associated facet in Δ is not marked. Marking, mark erasing and mark checking for a facet F_i in Δ can be done in constant time if we know the index i.

Example 4.1. Let Δ be the facet complex

$$\{F_1 = xz, F_2 = yz, F_3 = ztu, F_4 = twa, F_5 = uva, F_6 = ab\}$$

$\Delta =$



Then

$\mathrm{CB}_\Delta = \{\langle 1, \{2,3\}\rangle, \langle 2, \{1,3\}\rangle, \langle 3, \{1,2,4,5\}\rangle, \langle 4, \{3,5,6\}\rangle, \langle 5, \{3,4,6\}\rangle, \langle 6, \{4,5\}\rangle\}$
$\mathrm{CB}_\Delta^{F_3} = \{\langle 1, \emptyset\rangle, \langle 2, \emptyset\rangle, \langle 3, \emptyset\rangle, \langle 4, \{5,6\}\rangle, \langle 5, \{4,6\}\rangle, \langle 6, \{4,5\}\rangle\}$
$\Delta_{F_3}^{F_4,F_5} = \{F_4, F_5\}$ and $\mathrm{CB}_{\Delta_{F_3}^{F_4,F_5}} = \{\langle 4, \{5\}\rangle, \langle 5, \{4\}\rangle\}$.

We have $F_6 \notin \Delta_{F_3}^{F_4,F_5} = \{F_4, F_5\}$. Equivalently, we can mark F_6 in Δ and avoid to consider 6 in $\mathrm{CB}_\Delta^{F_3}$, the relation described by the incidence matrices is the same.

Algorithm 4.3 (Tree decision sparse algorithm).
Input: a connected facet complex $\Delta = \{F_1, \dots, F_l\}$ with n vertices.
Output: **True** if Δ is a tree, **False** otherwise.

1. Build CB_Δ.
2. For each i s.t. $F_i \in \Delta$
   (a) Build $\mathrm{CB}_\Delta^{F_i}$.
   (b) For each $\langle G, G'\rangle \in \{\langle G_j, G_k\rangle \mid G_j, G_k \in i\text{-th row in } \mathrm{CB}_\Delta \text{ s.t. } j < k\}$
      i. If $G \leqslant_{F_i} G'$ or $G' \leqslant_{F_i} G$, continue with the next couple.
      ii. Build $\mathrm{CB}_{\Delta_{F_i}^{G,G'}}$ from $\mathrm{CB}_\Delta^{F_i}$ by marking Δ.
      iii. If G and G' are connected outside F_i in $\Delta_{F_i}^{G,G'}$ return **False**.
      iv. Erase all marks in Δ.
3. Return **True**.

Note that we build $\mathrm{CB}_{\Delta_{F_i}^{G,G'}}$ without building $\Delta_{F_i}^{G,G'}$ by using the previously computed incidence matrices.

Let us describe the algorithms for the sub procedures and their costs when not trivial.

1. Building CB_Δ: for every element $F_i \in \Delta$ we check if any other element $G \in (\Delta - F)$ is connected to F_i. Cost is $O(l^2 v_{F_i})$.
2. Building $\mathrm{CB}_\Delta^{F_i}$ having CB_Δ: for every s-row of CB_Δ and every element in the row $(H_1, \dots, H_{d_{F_s}})$ we check if the intersection holds outside F_i. This check can be done for every H with time cost v_H, and the total cost is hence $l \sum_{j=1}^{d_{F_s}} v_{H_j}$.
3. The cost of step 2(b).i is trivially $O(v_{F_i})$
4. Building $\mathrm{CB}_{\Delta_{F_i}^{G,G'}}$ having CB_Δ and $\mathrm{CB}_\Delta^{F_i}$: we don't actually build $\mathrm{CB}_{\Delta_{F_i}^{G,G'}}$ but we use $\mathrm{CB}_{\Delta_{F_i}}$ marking the facets in Δ that does not appear in $\mathrm{CB}_{\Delta_{F_i}^{G,G'}}$. Instead of looking for a connected path in $\mathrm{CB}_{\Delta_{F_i}^{G,G'}}$ outside F_i, we look equivalently for a connected path in $\mathrm{CB}_{\Delta_{F_i}}$ outside F_i whose links are not marked in Δ.
   There are two cases, $G \cap G' = \emptyset$ and $G \cap G' \neq \emptyset$.
   $G \cap G' = \emptyset$: we have to mark the facets $H \in \Delta$ for which $H \cap F_i \neq \emptyset$. These are the d_{F_i} elements in i-th row of CB_Δ. Cost is d_{F_i}.
   $G \cap G' \neq \emptyset$: we have to mark the facets $H \in \Delta$ for which $H \cap F_i \neq G \cap G'$. These are the elements outside i-th row of CB_Δ ($E - d_{F_i}$ markings) and the elements in the i-th row for which $H \cap F_i \neq G \cap G'$. ($d_{F_i}$ checks at cost v_{F_i} and at most d_{F_i} markings) The cost is hence $O(d_{F_i} v_{F_i} + E)$.
   Total cost for step 2(b).ii is thus $O(d_{F_i} v_{F_i} + E)$.
5. To check if G and G' are connected in $\Delta_{F_i}^{G,G'}$ outside F_i: having $\mathrm{CB}_{\Delta_{F_i}^{G,G'}}$ the problem reduces to check connectedness in a graph with $O(E)$ edges, and that can be done with the well known Breadth First Search technique at cost $O(E)$.
   Alternative: connection check using $\mathrm{CB}_{\Delta_{F_i}^{G,G'}}$: We start from the elements in the G row of $\mathrm{CB}_{\Delta_{F_i}^{G,G'}}$ in a dequeue list L. For every unmarked H there, we check if it is G', in which case we return true, we mark H in the complex and add to L the elements in the H row of $\mathrm{CB}_{\Delta_{F_i}^{G,G'}}$. At most l merging

at unitary cost, at most E elements in L and E marking setting/checking. Total cost is hence $O(E)$.

This algorithm is a straightforward application of Theorem 2.3. Its complexity is as follows:

We build CB_Δ once (cost $O(lE)$), then for every $i \in \{1,\ldots,l\}$ we build $\mathrm{CB}_\Delta^{F_i}$ at cost $v_{F_i}E$ and we perform steps 2(b).i-iv $\binom{d_{F_i}}{2}$ times, at cost $O(d_{F_i}v_{F_i}+E)$ for every iteration. The dominant cost is this last step. Total complexity is hence

$$O\left(\sum_{i=1}^{l}(d_{F_i}^3 v_{F_i} + d_{F_i}^2 E)\right) = O\left(\sum_{i=1}^{l}\left(d_{F_i}^3 v_{F_i} + d_{F_i}^2 \sum_{s=1}^{l} d_{F_s}\right)\right)$$

– If the complex is not sparse ($v_{F_i} \approx n$, $d_{F_i} \approx l$ for every $i \in \{1,\ldots,l\}$) then Sparse Algorithm complexity is equal to Standard Algorithm complexity, $O\big(nl^4\big)$.
– If $d_{F_i}, v_{F_i} < \sqrt{l}$, a possible definition of sparseness for a complex, then Sparse Algorithm complexity is $O\left(l^3\sqrt{l}\right)$.

Some Statistics

The line 400/3 example is a sparse tree, and the line 400/40 example is an almost sparse tree. The star line examples are trees but they are not sparse. Permuted means that the facet are fed to the algorithm not in the “natural” order but after a random reshuffle. The R stands for the removal of useless facets optimization.

Table 4. Standard Alg./Sparse Alg. Comparison

Example	Standard	Standard+R	Sparse	Sparse+R
line 400-3	14.2s	4.3s	1.53s	0.01s
line 400-3 Permuted	12.8s	12.8s	0.86s	0.85s
line 400-40	115s	3.2s	100s	0.2s
line 400-40 Permuted	97s	89s	80s	80s
star-line 4/100	12.7s	3.7s	14.2s	4.5s
star-line 4/100 Permuted	12.4s	12.7s	14.1s	13.3s
star-line 10/100	300s	89s	318s	99s
star-line 10/100 Permuted	290s	279s	309s	296s

For sparse examples the sparse algorithm is clearly better than the standard algorithm; as expected, this is not the case when the sparseness is lacking. The useless facet removal optimization is very sensitive to facet ordering, but is very useful when the conditions are suitable.

Conclusions and Further Work

Large sparse trees are the hardest cases for our tree checking algorithm. The sparse algorithm looks very promising for these cases, especially from the point of view of its practical complexity.

We are working on the further optimization of the sparse algorithm. Using this algorithm for checking examples, we will work on Zheng’s conjecture. We will then compare the sparse algorithm and the algorithm based on Zheng’s conjecture performance for sparse complexes.

References

1. M. Caboara, S. Faridi, P. Selinger, Prototype implementation of tree algorithms, available at `http://www.dm.unipi.it/~caboara/Research`.
2. M. Caboara, S. Faridi, P. Selinger. *Simplicial cycles and the computation of simplicial trees.* Journal of Symbolic Computation, to appear.
3. CoCoA Team, *CoCoA: a system for doing Computations in Commutative Algebra*, available at `http://cocoa.dima.unige.it/`.
4. S. Faridi, *The facet ideal of a simplicial complex*, Manuscripta Mathematica 109 (2002), 159–174.
5. S. Faridi, *Cohen-Macaulay properties of square-free monomial ideals*, Journal of Combinatorial Theory, Series A, 109 (2005), no. 2, 299–329.
6. S. Faridi, *Simplicial trees are sequentially Cohen-Macaulay*, J. Pure and Applied Algebra 190 (2004), 121–136.
7. S. Faridi, *Monomial ideals via square-free monomial ideals*, Lecture Notes in Pure and Applied Mathematics, to appear.
8. A. Simis, W. Vasconcelos, R. Villarreal, *On the ideal theory of graphs*, J. Algebra 167 (1994), no. 2, 389–416.
9. R. Villarreal, *Cohen-Macaulay graphs*, Manuscripta Math. 66 (1990), no. 3, 277–293.
10. X. Zheng, *Homological properties of monomial ideals associated to quasi-trees and lattices*, Ph.D. thesis, Universität Duisburg-Essen, August 2004.