Computing the Face Lattice of a Polytope from its Vertex-Facet Incidences

Volker Kaibel and Marc E. Pfetsch [*]

June 7, 2001

Abstract

We give an algorithm that constructs the Hasse diagram of the face lattice of a convex polytope P from its vertex-facet incidences in time $\mathcal{O}(\min\{n,m\} \cdot \alpha \cdot \varphi)$, where n is the number of vertices, m is the number of facets, α is the number of vertex-facet incidences, and φ is the total number of faces of P. This improves results of Fukuda and Rosta [4], who described an algorithm for enumerating all faces of a d-polytope P in $\mathcal{O}\big(\min\{n,m\} \cdot d \cdot \varphi^2\big)$ steps. For simple or simplicial d-polytopes our algorithm can be specialized to run in time $\mathcal{O}(d \cdot \alpha \cdot \varphi)$. Furthermore, applications of the algorithm to other atomic lattices are discussed, e.g., to face lattices of oriented matroids.

Keywords: polytope, face lattice, enumeration, vertex-facet incidences, oriented matroid

MSC 2000: 68R05 68U05 52B11 68Q25 52C40

1 Introduction

Let P be a d-polytope, i.e., a d-dimensional bounded convex polyhedron. It is well-known that the set $\mathcal{F}$ of its faces (including $\varnothing$ and P itself), ordered by inclusion, is a graded, atomic, and coatomic lattice: the *face lattice* of P. In particular, each face can be identified with its set of vertices or the set of facets it is contained in. In this paper, a face is generally identified

[*]TU Berlin, Fakultät II, Institut für Mathematik, MA 6–2, Straße des 17. Juni 136, 10623 Berlin, {kaibel,pfetsch}@math.tu-berlin.de. The first author has been supported by a DFG Gerhard-Hess-Forschungsförderungspreis (Zi 475/2-3).

with its vertices. We define $\varphi := |\mathcal{F}|$ and denote by $\mathcal{L}$ the Hasse diagram (as an abstract graph) of the face lattice. Hence, $\mathcal{L}$ is a directed rooted acyclic graph whose nodes correspond to the elements of $\mathcal{F}$. If ℓ_H, ℓ_G are nodes in $\mathcal{L}$, and $H, G \in \mathcal{F}$ are the corresponding faces of P, then there is an arc (ℓ_H, ℓ_G) in $\mathcal{L}$ if and only if $H \subset G$ and $\dim(G) = \dim(H) + 1$.

The *combinatorial face lattice enumeration problem* is the following: given a vertex-facet incidence matrix of P (see Section 2 for a definition), construct the Hasse diagram $\mathcal{L}$ of the face lattice. By definition, $\mathcal{L}$ is unlabeled. However, it might be desired to label each node of $\mathcal{L}$ corresponding to a face F with the set of (indices of) vertices contained in F, the set of (indices of) facets containing F, or with the dimension of F.

Fukuda and Rosta [4] gave an algorithm for the combinatorial face lattice enumeration problem for d-polytopes P which runs in $\mathcal{O}\big(\min\{n, m\} \cdot d \cdot \varphi^2\big)$ time, where m is the number of facets and n is the number of vertices of P. Since φ can be exponential in n and m (consider the d-simplex, for instance) it is, however, desirable to have an algorithm whose running time depends only linearly on φ (and polynomially on n and m).

For the *geometric face lattice enumeration problem*, which asks for the face lattice of a polytope that is given by an inequality description, there are algorithms satisfying this requirement on the running time, see e.g. [3]. However, in our context no geometric data are available.

Ganter [6] described an algorithm which, given the incidences of atoms and coatoms of a general atomic and coatomic lattice, enumerates all elements of that lattice in lexicographic order, where each element is identified with the set of its atoms. Specialized to our situation, one obtains an algorithm that computes all vertex sets of faces of P in $\mathcal{O}(\min\{n, m\} \cdot \alpha \cdot \varphi)$ steps, where α is the number of vertex facet incidences of P. We have $d \cdot \max\{n, m\} \le \alpha \le n \cdot m$, in particular, α is bounded polynomially in n and m. However, this algorithm does not compute the inclusion relations between the faces, i.e., the edges of the Hasse diagram of the face lattice. Of course, once all (vertex sets of) faces are computed, one may construct the Hasse diagram in an obvious way afterwards, but this would require a number of steps which is quadratic in the total number φ of faces.

Inspired by Ganter's algorithm, we developed the (quite different) algorithm presented below, which computes the entire Hasse diagram in the same running time of $\mathcal{O}(\min\{n, m\} \cdot \alpha \cdot \varphi)$ (see Theorem 1). In our algorithm the vertex set of each face or the set of facets it is contained in, as well as its dimension, is readily available (or can be computed without increasing the asymptotic running time). Of course, this may increase the (output) storage requirements significantly.

Fukuda and Rosta [4] also considered the combinatorial face lattice enumeration problem for the special case of simple or simplicial polytopes. They presented an algorithm that computes the Hasse diagram of a simple or simplicial polytope in $\mathcal{O}(d \cdot \varphi)$ steps, provided that in addition to the vertex facet incidences a *good orientation* of the graph is given as input data, i.e., an acyclic orientation that induces precisely one sink on every non-empty face. Unfortunately, no polynomial time algorithm is known that computes a good orientation of a simple polytope P — neither if P is given by its vertex-facet incidences nor if it is specified by its whole face lattice.

For simple or simplicial polytopes, our algorithm can be specialized such that it computes the Hasse diagram of the face lattice in $\mathcal{O}(d \cdot \alpha \cdot \varphi)$ steps from the vertex facet incidences, where no good orientation is required (see Section 3.1).

In Section 2.1 we give a rough sketch of the algorithm, which is followed by a more detailed description in Sections 2.2, 2.3, and 2.4. In Section 2.5 we analyze the algorithm. In Section 3 we present a special version of the algorithm for simple or simplicial polytopes (Section 3.1), a variant that computes the k-skeleton (Section 3.2), as well as a modification that computes the face lattice of an oriented matroid from its signed (co-)circuits (Section 3.3).

For the basic properties of polytopes that are important in our context, we refer to Ziegler's book [7]. The few concepts from the theory of algorithms and data structures that play a role in the paper can be found in any corresponding textbook (e.g. in the one of Cormen, Leiserson, and Rivest [2]).

2 The Algorithm

Define m to be the number of facets and n the number of vertices of the d-polytope P. Let $A = (a_{fv}) \in \{0,1\}^{m \times n}$ be a *vertex facet incidence matrix* of P. Hence the facets of P can be identified with $F = \{1, \ldots, m\}$ and its vertices can be identified with $V = \{1, \ldots, n\}$, such that $a_{fv} = 1$ if facet f contains vertex v, and $a_{fv} = 0$ otherwise. Denote by α the number of vertex facet incidences, i.e., the number of ones in A. For $S \subseteq V$ define $\mathrm{F}(S) := \{f \in F : a_{fs} = 1 \text{ for all } s \in S\}$, the set of facets containing all vertices of S. For $T \subseteq F$ define $\mathrm{V}(T) := \{v \in V : a_{tv} = 1 \text{ for all } t \in T\}$, the set of vertices contained in all facets of T.

For $S \subseteq V$, the set $\overline{S} := \mathrm{V}(\mathrm{F}(S))$ is the (vertex set of) the smallest face of P containing S (in lattice theoretic terms, the *join* of the elements in S).

One easily checks that this defines a *closure map* on the subsets of V, i.e., for all $S, S' \subseteq V$ we have

$$S \subseteq \overline{S}, \qquad \overline{\overline{S}} = \overline{S}, \qquad S \subseteq S' \Rightarrow \overline{S} \subseteq \overline{S'}\,.$$

The faces of P correspond exactly to the *closed sets* (i.e., sets $S \subseteq V$ with $\overline{S} = S$) of V with respect to this closure map. Our algorithm crucially relies on the fact that closures can be computed fast (see Section 2.2).

2.1 The Skeleton of the Algorithm

The strategy is to build up the Hasse diagram $\mathcal{L}$ of the face lattice from bottom ($\varnothing$) to top (P). Consequently, $\mathcal{L}$ is initialized with the single face $\varnothing$, and then enlarged iteratively by adding out-neighbors of nodes that have already been constructed. We will say that a face has been *seen*, once its corresponding node in $\mathcal{L}$ has been constructed.

During the algorithm, we keep a set $\mathcal{Q}$ containing those faces that we have seen so far, but for which we have not yet inserted their out-arcs into the Hasse diagram. At each major step, we remove a face H from the set $\mathcal{Q}$ and construct the set $\mathcal{G}$ of all faces G with $H \subset G$ and $\dim(G) = \dim(H)+1$. For each face $G \in \mathcal{G}$ we check whether it has already been seen. If this is not the case, then a new node in $\mathcal{L}$ representing G is constructed, and G is added to $\mathcal{Q}$. In any case, an arc connecting the node corresponding to H to the node corresponding to G is inserted into $\mathcal{L}$.

In order to compute the set $\mathcal{G}$, we exploit the fact that $\mathcal{G}$ consists of the inclusion minimal faces among the ones that properly contain H. Since the face lattice of a polytope is atomic, each face $G \in \mathcal{G}$ must be of the form $H(v) := \overline{H \cup \{v\}}$ for some vertex (atom) v; in particular, the Hasse diagram has at most $n \cdot \varphi$ arcs. Thus, we first construct the collection $\mathcal{H}$ of all $H(v)$, $v \in V \setminus H$, and then compute $\mathcal{G}$ as the set of inclusion minimal sets of $\mathcal{H}$.

Computing $H(v)$ for some $v \in V \setminus H$ requires to determine a closure. In Section 2.2 we describe a method to perform this task in $\mathcal{O}(\alpha)$ steps. Determining the inclusion minimal sets in the collection $\mathcal{H}$ clearly could be done in $\mathcal{O}(n^3)$ steps by pairwise comparisons, since $\mathcal{H}$ has at most n elements. However, in Section 2.3 we show that this can even be performed in $\mathcal{O}(n^2)$ time.

In Section 2.4 we describe a data structure that allows us to locate the node in $\mathcal{L}$ representing a given face G, or to assert that G has not yet been seen so far in $\mathcal{O}(\alpha)$ steps.

A summary of the analysis of the time complexity of the algorithm, along with a pseudo-code description of it, is given in Section 2.5.

2.2 Computing Closures

In order to be able to compute closures fast, we store the incidence matrix A in a sparse format both in a row and column based way. For each vertex $v \in V$, the elements in $\mathrm{F}(\{v\})$ (a subset of $\{1, \ldots, m\}$) are sorted increasingly (in $\mathcal{O}(m)$ time, see [2, Chap. 9.2]) and stored in a sorted list. Similarly, for each facet $f \in F$, we store the sorted set $\mathrm{V}(\{f\})$ in a list. This can be done in $\mathcal{O}(n \cdot m)$ time (which is dominated by $\mathcal{O}(n \cdot \alpha)$ and thus does not influence the estimate of the asymptotic running time in Proposition 1). The sparse format uses $\mathcal{O}(\alpha)$ storage.

Whenever we want to compute the closure of a set $S \subseteq V$, the first step is to compute $\mathrm{F}(S)$, i.e., the intersection of the lists corresponding to the elements in S. Since the intersection of two sorted lists can be computed in time proportional to the sum of the lengths of the two lists, and because the intersection of two lists is at most as long as the shorter one, $\mathrm{F}(S)$ can be computed in time $\mathcal{O}\big(\sum_{v \in S} |\mathrm{F}(\{v\})|\big) \subseteq \mathcal{O}(\alpha)$. Similarly, $\mathrm{V}(T)$ can be computed in time $\mathcal{O}(\alpha)$ for a set $T \subseteq F$.

Lemma 1. *The closure $\overline{S}$ of a set $S \subseteq V$ can be computed in $\mathcal{O}(\alpha)$ steps (provided that the vertex facet incidence matrix is given in the sparse format).*

2.3 Identifying the Minimal Sets

Suppose that $H \subset V$ is a face of P and $\mathcal{H}$ is the collection of faces $H(v) = \overline{H \cup \{v\}} \subseteq V$, $v \in V \setminus H$. Notice that for $v, w \in V \setminus H$ with $w \in H(v)$, we have $H(w) \subseteq H(v)$, since $H(w)$ is the intersection of all faces containing H and w (one of which is $H(v)$).

Our procedure to identify the set $\mathcal{G}$ of minimal sets in the collection $\mathcal{H}$ starts by assigning a label *candidate* to each vertex in $V \setminus H$. Subsequently, the *candidate* label of each vertex will either be removed or replaced by a label *minimal.* We keep the following three invariants: for each vertex v labeled *minimal* we have $H(v) \in \mathcal{G}$; for two different vertices v and w both labeled *minimal* we have $H(v) \neq H(w)$; $\mathcal{G}$ is contained in the set of all $H(v)$ for v labeled *minimal* or *candidate.* Clearly, if no vertex is labeled *candidate* anymore, the set of vertices labeled *minimal* is in one-to-one correspondence to $\mathcal{G}$ via $H(\cdot)$.

Suppose there is still some *candidate* v available. If $H(v)$ contains some vertex w which is labeled *minimal* or *candidate*, then we remove the *candidate* label from v (because of $H(w) \subseteq H(v)$). Otherwise, we label v *minimal.*

It is guaranteed by induction that the three invariants are satisfied throughout the procedure. Moreover, at each major step (choosing a *candidate* v) the number of *candidate* labels decreases by one. Since each such step takes $\mathcal{O}(n)$ time, the entire procedure has complexity $\mathcal{O}(n^2)$.

Lemma 2. *The set $\mathcal{G}$ of inclusion minimal sets in the collection $\mathcal{H}$ can be identified in $\mathcal{O}(n^2)$ steps.*

2.4 Locating Nodes

In order to keep track of the faces that we have seen so far and their corresponding nodes in $\mathcal{L}$ we maintain a special data structure, the *face tree.* In this data structure, a face $S = \{s_1, \ldots, s_k\} \subseteq V$ (with $s_1 < \cdots < s_k$) is identified with the lexicographically smallest set $c(S) \subseteq S$ that generates S (i.e., $\overline{c(S)} = S$). The map $c(\cdot)$ is one-to-one; its inverse map is the closure map.

The set $c(S)$ can be computed as follows. For $1 \leq k \leq 2$ set $c(S) := S$. For $k \geq 3$, $c(S)$ is computed iteratively: initialize $c(S)$ with $\{s_1, s_2\}$; at each iteration extend $c(S)$ by the smallest s_i such that $\overline{c(S)} \subsetneq \overline{c(S) \cup \{s_i\}}$. Note that $|c(S)| \leq \dim(S) + 1 \leq d + 1$. Recall that we stored the vertex facet incidences in a sparse format (see Section 2.2). Thus, the whole computation can be performed in $\mathcal{O}(\alpha)$ steps, since just the intersections $\mathrm{F}(\{s_1\}) \cap \cdots \cap \mathrm{F}(\{s_i\})$, $i = 1, \ldots, k$, have to be computed iteratively. Then, $c(S)$ is obtained as the set of those s_i for which the intersection becomes smaller.

The arcs of the face tree are directed away from the root. They are labeled with vertex numbers, such that no two arcs leaving the same node have the same label and on every directed path in the tree the labels are increasing. Via the sets of labels on the paths from the root, the nodes of the tree correspond to the sorted sets $c(S)$ for the faces $S \subseteq V$ that have been seen so far. Each node has a pointer to the corresponding node of $\mathcal{L}$. By construction, the depth of the tree is bounded by $d + 1$.

Suppose that we want to find the node ℓ_S corresponding to some face $S \subseteq V$ in the part of $\mathcal{L}$ that we have already constructed or to assert that this face has not yet been seen so far. We first sort S (a subset of $\{1, \ldots, n\}$) in $\mathcal{O}(n)$ steps (see [2, Chap. 9.2]) and compute $c(S)$ in $\mathcal{O}(\alpha)$ steps. Then, starting from the root, we proceed (as long as possible) downwards in the face tree along arcs labeled by the successive elements of $c(S)$. Either we find an existing node in the tree which corresponds to S or we have to introduce new (labeled) arcs in the tree until we have constructed a node representing S.

In the latter case, it might be necessary to construct an entire new path in the tree. However, the definition of the representing sets $c(S)$ ensures that all "intermediate nodes" on that path will correspond to representing sets of faces as well. Hence, the number of nodes in the face tree always will be bounded by φ. The faces corresponding to intermediate nodes will be seen later in the algorithm, and consequently the corresponding pointer to $\mathcal{L}$ is set to `nil` for the meantime. Later in the algorithm, when we are searching for the face represented by such a tree-node for the first time, the `nil`-pointer will indicate that this face is not yet represented in $\mathcal{L}$. The `nil`-pointer is then replaced by a pointer to a newly created node representing the face in $\mathcal{L}$.

In any case, since the face tree has depth at most $d+1$ and the out-degree of each node is at most n, we need a total time of $\mathcal{O}(n + \alpha + (d+1) \cdot n) = \mathcal{O}(\alpha)$ to either locate resp. create the tree-node representing a certain face.

Lemma 3. *Using the face tree, it is possible to locate or create the node in $\mathcal{L}$ representing G in $\mathcal{O}(\alpha)$ steps (provided the vertex facet incidence matrix is stored in the sparse format).*

In the description given above we have assumed that for each node in the face tree the out-arcs are stored in a list which is searched linearly for a certain label when walking down the tree. Of course, one might store the set of out-arcs in a search tree (see, e.g., [2, Chap. 14]), allowing to perform the check for a certain label in logarithmic time. In practice, this might speed up the algorithm, while it does not improve the asymptotic running time.

2.5 The Analysis

We summarize the algorithm in pseudo-code below (Algorithm 1).

Proposition 1. *Algorithm 1 computes the Hasse diagram of the face lattice of a polytope P from its vertex facet incidences in $\mathcal{O}(n \cdot \alpha \cdot \varphi)$ time. It can be implemented such that its space requirements (without output space) are bounded by $\mathcal{O}(\varphi)$.*

Proof. Algorithm 1 works correctly by the discussion above.

Step 7 can be performed in $\mathcal{O}(n \cdot \alpha)$ steps by Lemma 1. Lemma 2 shows that we can do Step 8 in $\mathcal{O}(n^2) \subseteq \mathcal{O}(n \cdot \alpha)$ time. Hence Steps 7 and 8 in total contribute at most $\mathcal{O}(n \cdot \alpha \cdot \varphi)$ to the running time (since the while-loop is executed once per face).

The for-loop has to be executed for each of the $\mathcal{O}(n \cdot \varphi)$ arcs in the Hasse diagram $\mathcal{L}$. Lemma 3 implies that each execution of the for-loop can be performed in $\mathcal{O}(\alpha)$ steps. Thus, the claim on the running time follows.

Algorithm 1 Combinatorial enumeration of the face lattice of a polytope

```
1: **Input:** incidence matrix of a polytope $P$
2: **Output:** Hasse diagram $\mathcal{L}$ of the face lattice of $P$
3: initialize $\mathcal{L}$ and a face tree with $\ell_\varnothing$ corresponding to the empty face
4: initialize a set $\mathcal{Q} \subseteq \{\text{nodes of } \mathcal{L}\} \times \{\text{subsets of } V\}$ by $(\ell_\varnothing, \varnothing)$
5: while $\mathcal{Q} \neq \varnothing$ do
6:    choose some $(\ell_H, H) \in \mathcal{Q}$ and remove it from $\mathcal{Q}$
7:    compute the collection $\mathcal{H}$ of all $H(v)$, $v \in V \setminus H$
8:    compute the set $\mathcal{G}$ of minimal sets in $\mathcal{H}$
9:    for each $G \in \mathcal{G}$ do
10:      locate/create the node $\ell_G$ corresponding to $G$ in $\mathcal{L}$
11:      if $\ell_G$ was newly created then
12:         add $(\ell_G, G)$ to $\mathcal{Q}$
13:      end if
14:      add the arc $(\ell_H, \ell_G)$ to $\mathcal{L}$
15:   end for
16: end while
```

Since each node of the face tree corresponds to a face of P, the face tree has $\mathcal{O}(\varphi)$ nodes, and thus size $\mathcal{O}(\varphi)$. In order to guarantee that the space required for the storage of $\mathcal{Q}$ is bounded by $\mathcal{O}(\varphi)$ as well, one has to implement the algorithm slightly differently from its description above. Instead of storing the pairs (l_H, H) (each of which might require $\Omega(n)$ space), we just store pointers to the corresponding nodes in the face tree. The face tree is organized such that one can also walk towards its root. Then, in Step 6, we can compute the pair (l_H, H) from such a pointer in $\mathcal{O}(\alpha)$ time (walking upwards yields $c(H)$, computing $\overline{c(H)}$ yields H), which does not increase the total running time (asymptotically). □

If $m < n$, then it is more efficient to apply Algorithm 1 to the incidences of the dual polytope, i.e., to the transpose of the incidence matrix. Of course, after the computations one then has to exchange the roles of vertices and facets again. This yields the main result of the paper.

Theorem 1. *The Hasse diagram of the face lattice of a polytope P can be computed from the vertex facet incidences of P in $\mathcal{O}(\min\{n, m\} \cdot \alpha \cdot \varphi)$ time, where n is the number of vertices, m is the number of facets, α is the number of vertex facet incidences, and φ is the total number of faces of P. The space requirements of the algorithm (without output space) can be bounded by $\mathcal{O}(\varphi)$.*

Whenever a new node representing a face G in the Hasse diagram $\mathcal{L}$ is constructed, we can label that node with the vertex set of G, the set of facets containing G, or with the dimension of G without (asymptotically) increasing the running time of the algorithm. However, the output might become much larger due to such labelings. For instance, labeling the Hasse diagram of the d-cube by vertex labels requires $\Omega(4^d)$ output storage space, while the Hasse diagram with facet labels needs only $\mathcal{O}(d \cdot 3^d)$ space.

In practice, the computation can be speeded up by exploiting that every vertex contained in a face G with $H \subset G$ and $\dim G = \dim H + 1$ must be contained in some facet which contains H. Thus, it suffices to consider only the sets $H(v)$, $v \in \bigcup_{S \in \mathrm{F}(H)} S \setminus H$ in Step 7.

3 Extensions

3.1 Simple or Simplicial Polytopes

For a simple d-polytope P with n vertices, the above procedure can be implemented more efficiently. First observe that $\alpha = n \cdot d$ in this case. From the incidences (stored in sparse format), the graph $G(P)$ of P (i.e., all one-dimensional faces) can be computed in time $\mathcal{O}(n^2 \cdot d)$, since a pair of vertices forms an edge if and only if it is contained in $d-1$ common facets. After initialization with the vertices instead of $\varnothing$ (in Steps 3, 4), Step 7 can now be simplified. For some arbitrary vertex $v \in H$ we only need to check the at most d neighbors of v in $G(P)$ outside H. Each of these will yield an arc in the Hasse diagram, thus no non-minimal faces are constructed in Step 7. Hence, Step 8 can be skipped. The total running time for simple d-polytopes thus decreases to $\mathcal{O}(d \cdot \alpha \cdot \varphi)$ (since the for-loop is executed at most $d \cdot \varphi$ times). The space complexity stays $\mathcal{O}(\varphi)$ (without output space). By duality, the same running times and space requirements can be achieved for simplicial polytopes.

Similarly to the situation with general polytopes, for both simple and simplicial polytopes we can also output for each face its vertices, the facets containing it, or its dimension without (asymptotically) increasing the running time.

3.2 The k-Skeleton

A variant of the algorithm computes the Hasse diagram of the k-skeleton of a polytope P. One simply prevents the computation of faces of dimensions larger than k by not inserting any $(k-1)$-face into the list $\mathcal{Q}$. This leads to an $\mathcal{O}\big(n \cdot \alpha \cdot \varphi^{\leq k}\big)$ time algorithm, where $\varphi^{\leq k}$ is the number of faces of P of dimension at most k.

3.3 Oriented Matroids

Algorithm 1 can be applied to the enumeration of general atomic lattices with the property that one can compute the join of a set of atoms. For instance, this holds for every atomic and coatomic lattice if the atom-coatom incidences are given, because in this case one can compute the joins of atoms similarly to the case of face lattices of polytopes.

The (Edmonds-Mandel) face lattice of an oriented matroid (see Chapter 4 of [1] for definitions and background) is the set of all covectors plus an additional maximal element $\hat{1}$ (ordered by the usual partial order $\preceq$). This lattice is atomic and coatomic. Hence, we can compute its Hasse diagram from its abstract atom-coatom incidences as above.

However, this is not the usual way to encode an oriented matroid. It is more common to specify an oriented matroid by a set of sign-vectors from $\{-,0,+\}^k$, e.g., by its covectors. The cocircuits are the $\preceq$-minimal covectors of the oriented matroid, i.e., the atoms of its face lattice. The join of two covectors simply is their *composition*, if their *separation set* is empty, or $\hat{1}$ otherwise. Such a composition can be computed in $\mathcal{O}(k)$ steps, which enables us to compute the face lattice (efficiently) from its cocircuits by a variant of Algorithm 1.

If n denotes the number of cocircuits of $\mathcal{M}$, Step 7 can be performed in $\mathcal{O}(n \cdot k \cdot \varphi)$ steps altogether (where φ is the total number of covectors of $\mathcal{M}$). Step 8 takes $\mathcal{O}\big(n^2 \cdot k \cdot \varphi\big)$ time in total. The face tree is organized as for Algorithm 1. One fixes an ordering $C_1, \dots, C_n$ of the cocircuits. For a covector S let $\{i_1, \dots, i_r\}$ $(i_1 < \cdots < i_r)$ be the set of atoms below S in the face lattice. Then we iteratively form the joins of $C_{i_1}, \dots, C_{i_r}$, and let $c(S)$ consist of all those indices for which the "joins change." Computing $c(S)$ from S takes $\mathcal{O}(n \cdot k)$ steps.

Using this modified face tree, a given covector S can now be searched in the same way as in the case of face lattices of polytopes. The depth of the face tree is bounded by k. Hence, location/creation of a covector can be done in $\mathcal{O}(n \cdot k)$ time. The rest of the analysis is similar to the proof of Proposition 1. Thus, the Hasse diagram of the face lattice of an oriented matroid can be computed in $\mathcal{O}\big(n^2 \cdot k \cdot \varphi\big)$ steps.

The case where the *maximal* covectors (i.e., the coatoms) of an oriented matroid are given is a bit different. Here, the number of faces is bounded by m^2, where m is the number of maximal covectors. Hence, the size of the face lattice is polynomial in m. Fukuda, Saito, and Tamura [5] give an $\mathcal{O}(k^3 \cdot m^2)$ time algorithm for constructing the faces lattice from the maximal covectors.

Acknowledgements

We are indepted to Michael Joswig for stimulating discussions and to Günter M. Ziegler for valuable comments on the paper.

References

[1] A. Björner, M. Las Vergnas, B. Sturmfels, N. White, and G. M. Ziegler, *Oriented Matroids*, vol. 46 of Encyclopedia of Mathematics and its Applications, Cambridge University Press, 2nd ed., 1999.

[2] T. H. Corman, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, 1990.

[3] K. Fukuda, T. M. Liebling, and F. Margot, *Analysis of backtrack algorithms for listing all vertices and all faces of a convex polyhedron*, Comput. Geom., 8 (1997), pp. 1 – 12.

[4] K. Fukuda and V. Rosta, *Combinatorial face enumeration in convex polytopes*, Comput. Geom., 4 (4) (1994), pp. 191–198.

[5] K. Fukuda, S. Saito, and A. Tamura, *Combinatorial face enumeration in arrangements and oriented matroids*, Discrete Appl. Math., 31 (2) (1991), pp. 141 – 149.

[6] B. Ganter, *Algorithmen zur formalen Begriffsanalyse*, in Beiträge zur Begriffsanalyse, B. Ganter, R. Wille, and K. E. Wolff, eds., B.I. Wissenschaftsverlag, 1987, pp. 241–254.

[7] G. M. Ziegler, *Lectures on Polytopes*, Springer-Verlag, 1995. Revised edition 1998.