A FACE ITERATOR FOR POLYHEDRA AND
MORE GENERAL FINITE LOCALLY BRANCHED LATTICES

JONATHAN KLIEM AND CHRISTIAN STUMP

Abstract. We discuss a new memory-efficient depth-first algorithm and its implementation that iterates over all elements of a finite locally branched lattices. This algorithm can be applied to face lattices of polyhedra and various generalizations such as finite polyhedral complexes and subdivisions of manifolds, extended tight spans and closed sets of matroids. Its practical implementation is very fast compared to state-of-the-art implementations of previously considered algorithms.

1. Introduction

We call a finite lattice \((P, \leq)\) locally branched if all intervals of length 2 contain at least four elements. Such lattices are atomic and coatomic. We refer to Section 2 for details.

This paper describes a depth-first algorithm to visit exactly once all elements in a finite locally branched lattice given its coatoms, see Section 3. It moreover describes variants of this algorithm allowing the iteration over slightly more general posets. Most importantly, examples of such locally branched lattices (or its mild generalizations) include face posets of

- polytopes and unbounded polyhedra,
- finite polytopal or polyhedral complexes,
- finite polyhedral subdivisions of manifolds,
- extended tight spans, and
- closed sets of matroids.

One may in addition compute all cover relations, see Subsection 4.1. The provided theoretical runtime (without variants) is the same as of the algorithm discussed by V. Kaibel and M. E. Pfetsch in [4], see Section 4. In the slightly generalized situations, the theoretical runtime might be better as for extended tight spans (without chords) with many facets (using the opposite lattice), or might be worse as for extended tight spans with many vertices.

In practice it appears that the chosen data structures and implementation details make the implementation\(^1\) very fast for the iteration and still fast for cover relations in the graded case compared to state-of-the-art implementations of previously considered algorithms, see Section 5.

Acknowledgements. We thank Michael Joswig for valuable discussions and in particular for providing multiple relevant references. We further thank all participants of the trac ticket \#26887 for stimulating discussions.

Date: May 7, 2019.

J.K. receives funding by the Deutsche Forschungsgemeinschaft DFG under Germany’s Excellence Strategy – The Berlin Mathematics Research Center MATH+ (EXC-2046/1, project ID: 390685689). C.S. is supported by the DFG Heisenberg grant STU 563/4-1 “Noncrossing phenomena in Algebra and Geometry”.

\(^1\)See https://trac.sagemath.org/ticket/26887.
2. Formal framework

Let \((\mathcal{P}, \leq)\) be a finite poset and denote by \(\prec\) its cover relations. We usually write \(\mathcal{P}\) for \((\mathcal{P}, \leq)\) and write \(\mathcal{P}^{\text{op}}\) for the opposite lattice \((\mathcal{P}^{\text{op}}, \leq_{\text{op}})\) with \(b \leq_{\text{op}} a\) for \(a \leq b\). For \(a, b \in \mathcal{P}\) with \(a \leq b\) we denote the interval as \([a, b] = \{p \in \mathcal{P} \mid a \leq p \leq b\}\). If \(\mathcal{P}\) has a lower bound \(\hat{0}\), its \(\text{atoms}\) are the upper covers of the lower bound, \(\text{Atoms}(\mathcal{P}) = \{p \in \mathcal{P} \mid \hat{0} \prec p\}\)

and, for \(p \in \mathcal{P}\), we write \(\text{Atoms}(p) = \{a \in \text{Atoms}(\mathcal{P}) \mid p \geq a\}\) for the atoms below \(p\).

**Definition 2.1.** \(\mathcal{P}\) is locally branched if for every saturated chain \(a \prec b \prec c\) there exists an element \(d\) with \(a < d < c\). If this element is unique, then \(\mathcal{P}\) is said to have the \text{diamond property}.

The diamond property is a well-known property of face lattices of polytopes, see [7, Theorem 2.7(iii)]. The property of being locally branched has also appeared in the literature in contexts different from the present under the name \(2\)-thick lattices, see for example [1] and the references therein.

An obvious example of a locally branched lattice is the boolean lattice \(B_n\) given by all subsets of \(\{1, \ldots, n\}\) ordered by containment. We will later see that all such lattices are meet semi-sublattices of \(B_n\).

In the following, we assume \(\mathcal{P}\) to be a finite lattice with meet operation \(\wedge\), join operation \(\vee\), lower bound \(\hat{0}\) and upper bound \(\hat{1}\). We say that

- \(\mathcal{P}\) is \text{atomic} if all elements are joins of atoms,
- \(\mathcal{P}\) is \text{coatomic} if all elements are meets of coatoms,
- \(p \in \mathcal{P}\) is \text{join-irreducible} if \(p\) has a unique lower cover \(q \prec p\),
- \(p \in \mathcal{P}\) is \text{meet-irreducible} if \(p\) has a unique upper cover \(p \prec q\).

Atoms are join-irreducible and coatoms are meet-irreducible. The following classification of atomic and coatomic lattices is well-known.

**Lemma 2.2.** We have that

(i) \(\mathcal{P}\) is atomic if and only if the only join-irreducible elements are the atoms.

(ii) \(\mathcal{P}\) is coatomic if and only if the only meet-irreducible elements are the coatoms.

**Proof.** Let \(p, q \in \mathcal{P} \setminus \hat{0}\) with \(p \neq q\). If \(\mathcal{P}\) is atomic then \(\bigvee \text{Atoms}(p) = p \neq q = \bigvee \text{Atoms}(q)\) and, in particular, \(\text{Atoms}(p) \neq \text{Atoms}(q)\). If \(p\) is join-irreducible and \(q \prec p\) then \(\text{Atoms}(p) = \text{Atoms}(q)\). The first equivalence follows. The second equivalence is the first applied to \(\mathcal{P}^{\text{op}}\). \(\square\)

**Examples 2.3.** The face lattice of a polytope has the diamond property, it is atomic and coatomic, and every interval is again the face lattice of a polytope. The face lattice of an (unbounded) polyhedron might neither be atomic nor coatomic as witnessed by the face lattice of the positive orthant in \(\mathbb{R}^2\) with five faces.

The reason to introduce locally branched posets is the following relation of this notion to atomic and coatomic lattices, which has, to the best of our knowledge, not appeared in the literature.
Proposition 2.4. The following statements are equivalent:

(i) $P$ is locally branched,
(ii) every interval of $P$ is atomic,
(iii) every interval of $P$ is coatomic.

Proof. $P$ is locally branched if and only if $P^\text{op}$ is locally branched. Also, $P$ is atomic if and only if $P^\text{op}$ is coatomic. Hence, it suffices to show (i) $\iff$ (ii). Suppose $P$ is not locally branched. Then, there exist $p \prec x \prec q$ such that the interval $[p, q]$ contains exactly those three elements. Clearly, $[p, q]$ is not atomic. Now suppose $[p, q] \subseteq P$ is not atomic and $x$ is join-irreducible with unique lower cover $y$ with $p \prec y \prec x$. There exists $z \in [p, q]$ with $z \prec y$ and the interval $[z, x]$ contains exactly those three elements. □

Example 2.5. On the left an example of a non-graded locally branched lattice. On the right an example of an atomic, coatomic lattice, which is not locally branched as the interval between the two larger red elements contains only three elements.

Let $P$ be a finite locally branched poset with atoms $\{1, \ldots, n\}$. As we have seen that $P$ is atomic, it follows for all $p \in P$ that

$$p = \bigvee \text{Atoms}(p).$$

The following proposition underlines the importance of subset checks and of computing intersections to understand finite locally branched lattices.

Proposition 2.6. The relation $\leq$ and meet $\wedge$ in a finite locally branched lattice is given by

(i) $p \leq q \iff \text{Atoms}(p) \subseteq \text{Atoms}(q)$.
(ii) $p \wedge q = \bigvee (\text{Atoms}(p) \cap \text{Atoms}(q))$.

Proof.

(i) If $p \leq q$ then clearly $\text{Atoms}(p) \subseteq \text{Atoms}(q)$. On the other hand, if $\text{Atoms}(p) \subseteq \text{Atoms}(q)$, then $p = \bigvee \text{Atoms}(p) \leq \bigvee \text{Atoms}(q) = q$, as $\bigvee \text{Atoms}(q)$ is in particular an upper bound for $\text{Atoms}(p)$.

(ii) By (i) it holds that $\bigvee (\text{Atoms}(p) \cap \text{Atoms}(q))$ is a lower bound of $p$ and $q$. Also, $\text{Atoms}(p \wedge q) \subseteq \text{Atoms}(p)$ and we obtain $\text{Atoms}(p \wedge q) \subseteq \text{Atoms}(p) \cap \text{Atoms}(q)$. □

This proposition provides the following meet semi-lattice embedding of any finite locally branched lattice into a boolean lattice.

Corollary 2.7. Let $P$ be a finite locally branched lattice. The map $p \mapsto \text{Atoms}(p)$ is a meet semi-sublattice embedding of $P$ into the boolean lattice $B_n$.

Example 2.8. The above embedding does not need to be a join semi-sublattice embedding as witnessed by the face lattice of a square in $\mathbb{R}^2$. 
**Remark 2.9.** Proposition 2.6 shows that checking whether the relation \( p \leq q \) holds in \( \mathcal{P} \) is algorithmically a subset check \( \text{Atoms}(p) \subseteq \text{Atoms}(q) \), while computing the meet is given by computing the intersection \( \text{Atoms}(p) \cap \text{Atoms}(q) \).

Justified by Corollary 2.7, we restrict our attention in this paper to meet semi-sublattices of the boolean lattice.

2.1. **Variants of this framework and examples.** Before presenting in Section 3 the algorithm to iterate over the elements of a finite locally branched lattice together with variants to avoid any element above certain atoms and to avoid any element below certain coatoms (or other elements of \( B_n \)), we give the following main use cases for such an iterator.

**Example 2.10 (Polytope).** The face lattice of a polytope \( P \) has the diamond property and is thus locally branched.

**Example 2.11 (Polyhedron).** The face lattice of a polyhedron \( P \) is isomorphic to the one obtained from quotiening out the affine space \( P \) contains. Thus, we can assume that \( P \) does not contain an affine line. It is well known (see e.g. [7, Exercise 2.19]) that we may add an extra facet \( \overline{F} \) to obtain a polytope \( \overline{P} \). The faces of \( P \) are exactly the faces of \( \overline{P} \) not contained in \( F \) (together with the empty face). Thus, the iterator visits all non-empty faces of \( P \) by visiting all faces of \( \overline{P} \) not contained in \( F \).

**Example 2.12 (Polytopal subdivision of manifold).** The face poset of a finite polytopal subdivision of a closed manifold (compact manifold without boundary). Adding an artificial upper bound \( \hat{1} \), this is a finite locally branched lattice.

**Example 2.13 (Extended tight spans).** We consider extended tight spans as defined in [3, Section 3] as follows: Let \( P \subset \mathbb{R}^d \) be a finite point configuration, and let \( \Sigma \) be a polytopal complex with vertices \( P \), which covers the convex hull of \( P \). We call the maximal cells of \( \Sigma \) facets. We can obtain a closed \( d \)-manifold \( M \) from \( \Sigma \) by adding facets. In most cases one facet containing all vertices of the boundary of \( \Sigma \) will suffice. When \( \Sigma \) contains few facets or a chord, we might have to add one vertex at infinity and for each face \( F \) in the boundary of \( \Sigma \) a face \( F \cup \{\infty\} \). Now we have embedded the face poset of \( \Sigma \) into the face lattice with \( \hat{1} \) of a \( d \)-manifold \( M \). Given a collection \( \Gamma \) of boundary faces of \( \Sigma \). An element of \( \Sigma \) has empty intersection with all elements of \( \Gamma \), if it does not contain any vertex of one of the elements in \( \Gamma \). We can iterate over all elements of \( \Sigma \), which do not contain any vertex in \( V(\Gamma) \), all the vertices of \( \Gamma \).

**Example 2.14 (Closed sets of a matroid).** The MacLane–Steinitz exchange property (see e.g. [5, Lemma 1.4.2]) ensures that the closed sets of a matroid form a locally branched finite lattice.

**Example 2.15 (Locally branched lattices with non-trivial intersection).** Let \( \mathcal{P}_1, \ldots, \mathcal{P}_k \) be finite locally branched meet semi-sublattices of \( B_n \). Then the iterator may iterate through all elements of their union by first iterating through \( \mathcal{P}_1 \), then through all elements in \( \mathcal{P}_2 \) not contained in \( \mathcal{P}_1 \) and so on.

**Example 2.16 (Polyhedral complexes).** Using the iteration as in the previous example allows to iterate through polytopal or polyhedral complexes, or through complexes of tight spans.
3. The algorithm

Let $\mathcal{P}$ be a finite locally branched lattice given as a meet semi-sublattice of the boolean lattice $B_n$. This is, $\text{Atoms } \mathcal{P} = \{1, \ldots, n\}$ and $p = \text{Atoms}(p) \in \mathcal{P}$. The following algorithm is a recursively defined depth-first iterator through the elements of $\mathcal{P}$. Given $c \in \mathcal{P}$ and its lower covers $x_1, \ldots, x_k$, the iterator yields $c$ and then computes, one after the other, the lower covers of $x_1, \ldots, x_k$, taking into account those to be ignored, and then recursively proceeds. Being an iterator means that the algorithm starts with only assigning the input to the respective variables and then waits in its current state. Whenever an output is requested, it starts from its current state and runs to the point \textbf{ITERATOR OUTPUT}, outputs the given output, and again waits. Iterators are regularly used in modern programming languages.\footnote{See https://en.wikipedia.org/wiki/Iterator.}

A slightly more sophisticated version of this algorithm is implemented in SageMath\textsuperscript{1}. Before proving the correctness of the algorithm, we provide several detailed examples. If not mentioned otherwise, we do not ignore any atoms and always set $\text{ignored}_\text{atoms} = \emptyset$ in the examples. We also assume the lists to be ordered lexicographically for iteration. One may assume that the algorithm additionally visits the upper bound given by the union of the coatoms, whenever this is suitable.

Example 3.1 (Square). We apply the algorithm to visit faces of a square.

\begin{itemize}
  \item \textbf{INPUT}: $\text{coatoms} = \left\{ \{1,2\}, \{1,4\}, \{2,3\}, \{3,4\} \right\}$, $\text{ignored}_\text{sets} = \emptyset$
  \item $a = \{1,2\}$, \textbf{ITERATOR OUTPUT}: $\{1,2\}$
\end{itemize}

\begin{algorithm}
\begin{algorithmic}[1]
  \STATE \textbf{Algorithm \text{FaceIterator}}
  \STATE \textbf{INPUT}
  \STATE \quad \bullet \text{coatoms} -- list of coatoms of $\mathcal{P}$ not contained in any of $\text{ignored}_\text{sets}$
  \STATE \quad \bullet \text{ignored}_\text{sets} -- list of subsets of $\{1, \ldots, n\}$
  \STATE \quad \bullet \text{ignored}_\text{atoms} -- subset of $\{1, \ldots, n\}$
  \STATE \textbf{PROCEDURE}
  \STATE \quad \textbf{if} \ \text{coatoms} \neq \emptyset :
  \STATE \quad \quad a := \text{coatoms} . \text{first}_\text{element} ()
  \STATE \quad \quad \textbf{if} \ a \cap \text{ignored}_\text{atoms} = \emptyset :
  \STATE \quad \quad \quad \textbf{ITERATOR OUTPUT}
  \STATE \quad \quad \quad \bullet \ a
  \STATE \quad \quad \quad \textbf{new}_\text{coatoms} = \{ \ a \cap b : b \in \text{coatoms} \ \setminus \ a \ \}
  \STATE \quad \quad \textbf{new}_\text{coatoms} = \{ \ x \in \text{new}_\text{coatoms} : x \not\subseteq y \ \text{for all} \ y \in \text{ignored}_\text{sets} \ \}
  \STATE \quad \quad \textbf{FaceIterator} ( \ \text{coatoms} = \text{new}_\text{coatoms} ,
  \quad \quad \text{ignored}_\text{sets} = \text{ignored}_\text{sets} . \text{copy} () ,
  \quad \quad \text{ignored}_\text{atoms} = \text{ignored}_\text{atoms} )
  \STATE \quad \textbf{next}_\text{coatoms} = \{ \ x \in \text{coatoms} : x \not\subseteq a \cup \text{ignored}_\text{atoms} \ \}
  \STATE \quad \textbf{ignored}_\text{sets} . \text{append} ( a \cup \text{ignored}_\text{atoms} )
  \STATE \quad \textbf{FaceIterator} ( \ \text{coatoms} = \text{next}_\text{coatoms} ,
  \quad \text{ignored}_\text{sets} = \text{ignored}_\text{sets} ,
  \quad \text{ignored}_\text{atoms} = \text{ignored}_\text{atoms} )
\end{algorithmic}
\end{algorithm}
• new_coatoms = [{1}, {2}]
• Apply FaceIterator to sublattice [0, {1, 2}]
  – INPUT: coatoms = [{1}, {2}], ignored_sets = []
  – a = {1}, ITERATOR OUTPUT: {1}
  – new_coatoms = [0]
• Apply FaceIterator to sublattice [0, {1}]
  * INPUT: coatoms = [0], ignored_sets = []
  * a = ∅, ITERATOR OUTPUT: ∅
  * (new_coatoms is empty)
  * Apply FaceIterator to sublattice [0, 0] without output
  * Add ∅ to ignored_sets (to the copy in this call of FaceIterator)
  * Reapply FaceIterator to sublattice [0, {1}]
  * INPUT: coatoms = [0], ignored_sets = [0]
    – ignored_sets = [1]  
    – Reapply FaceIterator to sublattice [0, {1, 2}]
      – INPUT: coatoms = [{2}], ignored_sets = [{1}]
      – a = {2}, ITERATOR OUTPUT: {2}
      – Apply FaceIterator to sublattice [0, {2}]
        * INPUT: coatoms = [], ignored_sets = [{1}]
        – ignored_sets = [{1}, {2}]
        – Reapply FaceIterator to sublattice [0, {1, 2}]
          – INPUT: coatoms = [], ignored_sets = [{1}, {2}]
• ignored_sets = [{1, 2}]
• Reapply FaceIterator to entire lattice
• INPUT: coatoms = [{1, 4}, {2, 3}, {3, 4}], ignored_sets = [{1, 2}]
• a = {1, 4}, ITERATOR OUTPUT: {1, 4}
• Apply FaceIterator to sublattice [0, {1, 4}]
  – INPUT: coatoms = [{4}], ignored_sets = [{1, 2}]
  – a = {4}, ITERATOR OUTPUT: {4}
  – Apply FaceIterator to sublattice [0, {4}] without output
• ignored_sets = [{4}, {1, 4}]
• ... further outputs: {2, 3}, {3}, {3, 4}

Example 3.2 (Minimal triangulation of \( \mathbb{RP}^2 \)).
Example 3.3 (Tight span).

- **INPUT**: coatoms = \([\{1, 2, 3, 4\}, \{1, 2, 5, 6\}, \{1, 3, 6\}, \{2, 4, 5\}]\), ignored_sets = \([\{3, 4, 5, 6\}]\), ignored_atoms = \(\{3, 4, 5, 6\}\)
- **a** = \(\{1, 2, 3, 4\}\), no output of \(\{1, 2, 3, 4\}\)
- **new_coatoms** = \(\{\{1, 2\}, \{1, 3\}, \{2, 4\}\}\)
  - **INPUT**: coatoms = \(\{\{1, 2\}, \{1, 3\}, \{2, 4\}\}\), ignored_sets = \(\{\{3, 4, 5, 6\}\}\), ignored_atoms = \(\{3, 4, 5, 6\}\)
  - **a** = \(\{1, 2\}\), **ITERATOR OUTPUT**: \(\{1, 2\}\), \(\{1\}\), \(\{2\}\)
  - ignored_sets = \(\{\{3, 4, 5, 6\}, \{1, 2, 3, 4, 5, 6\}\}, \text{ coatoms } = \{\}\)
- **ignored_sets** = \(\{\{3, 4, 5, 6\}, \{1, 2, 3, 4, 5, 6\}\}, \text{ coatoms } = \{\}\)

Example 3.4 (Polytopal complex).

\[\text{Incorrect application} \text{ by applying to all polyhedra in the complex:} \]
- **INPUT**: coatoms = \(\{(W, N, 0), \{N, E, 0\}, \{S, E, 0\}\}\), ignored_sets = \(\[]\)
- **a** = \(\{W, N, 0\}\), **ITERATOR OUTPUT**: \(\{W, N, 0\}\)
- **new_coatoms** = \(\{\{N, 0\}\}\), **ITERATOR OUTPUT**: \(\{N, 0\}\)
- **ignored_sets** = \(\{\{W, N, 0\}\}\)
- **a** = \(\{N, E, 0\}\), **ITERATOR OUTPUT**: \(\{N, E, 0\}\)
- **new_coatoms** = \(\{\{E, 0\}\}\), **ITERATOR OUTPUT**: \(\{E, 0\}\)
- **ignored_sets** = \(\{\{W, N, 0\}, \{N, E, 0\}\}\)
• \(a = \{S, E, 0\}\), ITERATOR OUTPUT: \(\{S, E, 0\}\)
• new_coatoms = []

Correct application by applying successively to all faces of all polyhedra:

• ITERATOR OUTPUT: \(\{W, N, 0\}\) (output of \(\hat{1}\) before applying FaceIterator)
  • Apply algorithm for \(\{W, N, 0\}\):
    – INPUT: coatoms = \([\{W, 0\}, \{N, 0\}\], ignored_sets = \([\{W, N\}\]
    – ITERATOR OUTPUT: \(\{W, 0\}, \{0\}, \{N, 0\}\)
  • ITERATOR OUTPUT: \(\{N, E, 0\}\)
  • Apply algorithm for \(\{N, E, 0\}\):
    – INPUT: coatoms = \([\{E, 0\}\], ignored_sets = \([\{W, N, 0\}, \{N, E\}\]
    – ITERATOR OUTPUT: \(\{E, 0\}\)
  • ITERATOR OUTPUT: \(\{S, E, 0\}\)
  • Apply algorithm for \(\{S, E, 0\}\):
    – INPUT: coatoms = \([\{S\}\], ignored_sets = \([\{W, N, 0\}, \{N, E, 0\}, \{S, E\}\]
    – ITERATOR OUTPUT: \(\{S, 0\}\)

3.1. Correctness of the algorithm. As assumed, let \(P\) be a locally branched meet semi-sublattice of the boolean lattice \(B_n\). In the following properties and their proofs, we indeed see that, if \(P\) is any meet semi-sublattice of \(B_n\), the algorithm surely visits an element \(p \in P\) not contained in any of ignored_sets and not containing any of ignored_atoms if the interval \([p, \hat{1}]\) is locally branched. If this interval is not locally branched, then \(p\) might or might not be visited.

**Proposition 3.5.** The algorithm FaceIterator is well-defined in the following sense: Let \(a \in\) coatoms. Then

(i) The call of FaceIterator in line 18 applies the algorithm to the sublattice \([\hat{0}, a]\).

(ii) The call of FaceIterator in line 24 applies the algorithm to \(P\) with \(a \cup \) ignored_atoms appended to ignored_sets and all coatoms contained in \(a \cup \) ignored_atoms removed.

**Proof.** The proof of (ii) is obvious. To prove (i), we have to show that the construction of new_coatoms in lines 15–17 is correct. First, observe that all elements in new_coatoms are strictly below the element \(a\). Next, let \(x < a < \hat{1}\) in \(P\). Since \(P\) is locally branched there is an element \(b \neq a\) with \(x < b < \hat{1}\), implying \(x = a \cap b\). If \(x\) is not contained in any element in ignored_sets, then the same holds for \(b\) and thus, \(b \in\) coatoms and \(x \in\) new_coatoms. This implies that new_coatoms are exactly the lower covers of \(a\) not contained in an element of ignored_sets, as desired. \(\square\)

**Theorem 3.6.** The algorithm FaceIterator iterates exactly once over all element in \(P\) which are not contained in any subset of ignored_sets, and does contain any element in ignored_atoms.

**Proof.** We argue by induction on the cardinality of coatoms. First note that the cardinalities of new_coatoms and next_coatoms in the two subsequent calls of FaceIterator in lines 18 and 24 are both strictly smaller than the cardinality of coatoms. If coatoms = \(\emptyset\), then all elements of \(P \setminus \hat{1}\) are contained in elements of ignored_sets, and the algorithm does correctly not output any element. Suppose that coatoms \(\neq \emptyset\) and let \(a\) be its first element assigned in line 10. Let \(p \in P\). If \(p\) is contained in an element of ignored_sets or contains an element in ignored_atoms then it is not outputed by the algorithm. Otherwise,
• if $p = a$, then the algorithm outputs $p$ correctly in line 13,
• if $p < a$, then $p$ is outputed in the call of \texttt{FaceIterator} in line 18 by induction,
• if $p \not\leq a$, then $p \not\leq a \cup \text{ignored\_atoms}$ and $p$ is outputed in the call of \texttt{FaceIterator} in line 24 by induction.

3.2. Variants of the algorithm. We finish this section with a dualization property followed by explicitly stating the result when applying the algorithm for the variants discussed in Subsection 2.1.

Corollary 3.7. If $\text{ignored\_sets}$ is a list of coatoms, then the algorithm can also be applied to $\mathcal{P}^{\text{op}}$ with the roles of atoms and coatoms interchanged.

We later see in Theorem 4.1 that considering $\mathcal{P}^{\text{op}}$ instead of $\mathcal{P}$ might be faster as the runtimes depends on the number of coatoms. For example, in Example 3.2 one could apply the algorithm to $\mathcal{P}^{\text{op}}$ to improve runtime as there are 10 facets but only 6 vertices.

Corollary 3.8. Let $P$ be a polytope. Provided the vertex-facet incidences of $\mathcal{P}$, the above algorithm visits exactly all faces of $P$ once.

Corollary 3.9. Let $P$ be an unbounded polyhedron and let $\overline{P}$ be a projectively equivalent polytope with marked face. Provided the vertex-facet incidences of $\overline{P}$, the above algorithm visits exactly all non-empty faces of $P$ once.

Actually, an intersection of two faces of $P$ is not contained in the marked facet at infinity, unless it is empty. Hence, one could even use the algorithm without providing the marked facet at infinity. This might or might not visit the empty face.

Corollary 3.10. Let $P$ be a finite polytopal subdivision of a closed manifold. Provided the vertex-facet incidences the above algorithm visits exactly all faces of $P$ once.

Corollary 3.11. Let $\Sigma$ be an extended tight span in $\mathbb{R}^d$ as described in Example 2.13. Let $V$ be a subset of vertices of $\Sigma$. Provided the facets and the boundary faces in their vertex description, the above algorithm visits exactly all faces of $\Sigma$ not containing any vertex of $V$.

Corollary 3.12. Let $P$ be a polyhedral complex. Provided each maximal cell as vertex-facet incidences with possibly marked face at infinity. The algorithm can be applied to visit each element in $P$ exactly once.

4. Data structures, memory usage, and theoretical runtime

The operations used in the algorithm are intersections, subset checks and unions. It will turn out that the crucial operation for the runtime is the subset check.

For the theoretical runtime we consider representation as (sparse) sorted-lists-of-atoms. However, in the implementation we use (dense) atom-incidence-bit-vectors. This is theoretically slightly slower, but the crucial operations can all be done using bitwise operations.

Observe that a sorted-lists-of-atoms needs as much memory as there are incidences. Consider two sets $A$ and $B$ (of integers) of lengths $a$ and $b$, respectively, and a (possibly unsorted) list $C$ of $m$ sets of total length $\alpha$. Using standard implementations, we assume in the runtime analysis that

• finding (and possibly deleting) a given element $x$ inside $C$ has runtime $O(\alpha)$,
• deleting all duplicates in $C$ has runtime $O(\alpha \cdot m)$,
intersection \( A \cap B \) and union \( A \cup B \) have runtime \( \mathcal{O}(\max(a,b)) \),
a subset check \( A \subseteq B \) has runtime \( \mathcal{O}(b) \) and
to check whether \( A \) is subset of any element in \( C \) has runtime \( \mathcal{O}(\alpha) \).

Let \( r + 1 \) be the number of elements in a longest chain in \( \mathcal{P} \), let \( m = |\text{coatoms}| \), and let
\[
\alpha = \sum_{a \in \text{coatoms} \cup \text{ignored_sets}} |a \cup \text{ignored_atoms}|.
\]
(In the case that \( \text{ignored_sets} \) and \( \text{ignored_atoms} \) are both empty, \( \alpha \) is the total number of atom-coatom incidences.) Moreover, let \( \varphi \) be the number of recursive calls of the algorithm. (In the case that \( \text{ignored_atoms} \) is empty, \( \varphi \) is the cardinality of the output of the algorithm. Otherwise, it is bounded by this cardinality.)

**Theorem 4.1.** The algorithm has memory consumption \( \mathcal{O}(\alpha \cdot r) \) and runtime \( \mathcal{O}(\alpha \cdot m \cdot \varphi) \).

**Proof.** Note first that at each recursive call of \texttt{FaceIterator} the number of \texttt{coatoms} is bounded by \( m \) and the total length of \texttt{coatoms}, \texttt{ignored_sets} and \texttt{ignored_atoms} is bounded each by \( \alpha \).

With above assumptions, \texttt{FaceIterator} has runtime \( \mathcal{O}(\alpha \cdot m) \) not considering recursive calls.

A single call of \texttt{FaceIterator} has memory usage at most \( c \cdot \alpha \) for a global constant \( c \), not taking into account the recursive calls. The call in line 24 does not need extra memory as all old variables can be discarded. The longest chain of the lattice \([0,a]\) is at most of length \( r - 1 \). By induction the call of \texttt{FaceIterator} in line 18 has total memory consumption at most \( (r-1) \cdot c \cdot \alpha \). The claimed bounds follow. \( \square \)

### 4.1. Computing all cover relations.

Applying the algorithm to a graded locally branched meet semi-sublattice of \( B_n \) while keeping track of the recursion depths allows an a posteriori sorting of the output by the level sets of the grading. The recursion depth is the number of iterative calls using line 18. We obtain the same bound for generating all cover relations as V. Kaibel and M. E. Pfetsch [4]. For this we additionally assume that

- a list of \( \phi \) sets each of length at most \( n \) can be sorted in time \( \mathcal{O}(n \cdot \phi \cdot \log \phi) \) and
- a set with \( a \) elements can be looked up in a sorted list of \( \phi \) sets in time \( \mathcal{O}(a \cdot \log \phi) \).

**Proposition 4.2.** Let \( \mathcal{P} \) be a graded meet sublattice of \( B_n \). Denote by \( \alpha \) the total length of the \texttt{coatoms} of \( \mathcal{P} \), by \( n \) the number of atoms, by \( m \) the number of \texttt{coatoms} and by \( \varphi \) the number of elements in \( \mathcal{P} \).

Assume each level set of \( \mathcal{P} \) to be given as sorted-lists-of-atoms, one can generate all cover relations in time \( \mathcal{O}(\alpha \cdot \min(m,n) \cdot \varphi) \).

**Proof.** First, we sort all level-sets. Then, we intersect each element with each coatom, obtaining its lower covers among other elements. We look up each intersection, to determine, which are lower covers.

Sorting the level sets is done in time \( \mathcal{O}(n \cdot \varphi \cdot \log \varphi) \). All such intersection are obtained in time \( \mathcal{O}(\varphi \cdot m \cdot n) \). For a fixed element the total length of its intersection with all coatoms is bounded by \( \alpha \). Hence, all lookups are done in time \( \mathcal{O}(\varphi \cdot \alpha \cdot \log \varphi) \).

Finally, we note that \( m, n \leq \alpha \) and that \( \log \varphi \leq \min(m,n) \). \( \square \)

In the general case, one first sorts all elements in \( \mathcal{P} \), and then intersects each element \( p \) with all coatoms. The inclusion maximal elements among those strictly below \( p \) are lower covers of \( p \). They can be looked up in the list of sorted elements to obtain an index. Observe that all this is done time \( \mathcal{O}(\alpha \cdot m \cdot \phi) \).
5. Performance of the Algorithm When Implemented in SageMath

We present running times for the several computations. These are performed on an Intel® Core™ i7-7700 CPU @ 3.60GHz x86_64-processor with 30 GB of RAM. The computations are done either using polymake 3.3 [2] or the presented algorithm using SageMath 8.6 with trac ticket # 26887 [6]. The default algorithms in SageMath 8.6 performs much worse than both and is not considered here. We computed

1. cover relations and f-vector with polymake from vertex-facet incidences,
2. f-vector with polymake using its default algorithm from standard polytope constructor,
3. all cover relations with SageMath,
4. f-vector with SageMath.

![Figure 1. Comparison of the runtimes. Every dot represents one best-of-five computation, and every shifted diagonal is a factor-10 runtime difference. Dots at the boundary represent Memory Overflow.](image-url)
Observe that the computation of the $f$-vector with polymake from vertex-facet incidences also calculates all cover relations.

The implementations in SageMath using bitwise operations also allow to additionally use instructions for intrinsics such as Advanced Vector Extensions$^3$. Those instructions allow comparison of up to 256 instead of 64 bits (representing atom-incidences) at once. This results in a runtime improvement of at least a factor 2 compared to the computations of the $f$-vector (4) We have omitted these additional computations here.

For every algorithm we record the best-of-five computation on
- the simplex of dimension $n$,
- several instances of the cyclic polyhedron of dimension 10 and 20,
- the associahedron of dimension $n$,
- the permutoahedron of dimension $n$ embedded in dimension $n + 1$,
- a 20-dimensional counterexample to the Hirsch-conjecture,
- the cross-polytope of dimension $n$,
- the Birkhoff-polytope of dimension $(n - 1)^2$,
- joins of such polytopes with their duals,
- Lawrence polytopes of such polytopes.

References

[1] Margaret M. Bayer and Gábor Hetyei, Generalizations of Eulerian partially ordered sets, flag numbers, and the Möbius function, Discrete Math. 256 (2002), no. 3, 577–593.
[2] Ewgenij Gawrilow and Michael Joswig, polymake: a framework for analyzing convex polytopes, Polytopes—combinatorics and computation (Oberwolfach, 1997), DMV Sem., vol. 29, Birkhäuser, Basel, 2000, pp. 43–73. MR 1785292
[3] Simon Hampe, Michael Joswig, and Benjamin Schröter, Algorithms for tight spans and tropical linear spaces, Journal of Symbolic Computation 91 (2019), 116 – 128, MEGA 2017, Effective Methods in Algebraic Geometry, Nice (France), June 12-16, 2017.
[4] Volker Kaibel and Marc E. Pfetsch, Computing the face lattice of a polytope from its vertex-facet incidences, Computational Geometry 23 (2002), 281–290.
[5] James G. Oxley, Matroid theory, Oxford University Press, Inc., New York, NY, USA, 1992.
[6] The Sage Developers, Sagemath, the Sage Mathematics Software System (Version 8.7.beta4), 2019, https://www.sagemath.org.
[7] Günter M. Ziegler, Lectures on polytopes., 7 ed., vol. 152, Berlin: Springer-Verlag, 2007 (English).

Appendix A. Detailed runtimes

We give, for each of the four computations, an example of how it is executed for the 2-simplex. Compute cover relations and $f$-vector in polymake from vertex-facet-incidences:

```
polytope > new Polytope (VERTICES_IN_FACETS =>
[[0,1],[0,2],[1,2]]) -> F_VECTOR;
```

Compute $f$-vector in polymake using the standard constructor:

```
polytope > simplex(2) -> F_VECTOR;
```

Compute all cover relations in SageMath:

$^3$See https://en.wikipedia.org/wiki/Advanced_Vector_Extensions.
sage: CombinatorialPolyhedron([[0,1],[0,2],[1,2]]).face_lattice_cover_relations()

Compute $f$-vector in SageMath:

sage: CombinatorialPolyhedron([[0,1],[0,2],[1,2]]).f_vector()

The runtimes of the four best-of-five computations on the various examples are as follows. MOF indicates that the process was killed due to Memory Overflow.

|              | (1) Time in s | (2) Time in s | (3) Time in s | (4) Time in s | (1) Time in s | (2) Time in s | (3) Time in s | (4) Time in s |
|--------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|
| 13-simp      | 1.6e-1        | 7.3e-2        | 7.0e-4        |               | 6-assoc       | 6.0e-2        | 1.0e-1        | 2.7e-2        | 3.9e-4        |
| 15-simp      | 3.7e-1        | 1.7e-1        | 1.4e-3        |               | 7-assoc       | 1.0e+0        | 5.6e-1        | 2.4e-1        | 2.9e-3        |
| 16-simp      | 8.6e-1        | 3.9e-1        | 2.7e-3        |               | 8-assoc       | 2.1e+1        | 4.2e+0        | 2.4e+0        | 3.2e-2        |
| 17-simp      | 2.1e+0        | 8.9e-1        | 5.5e-3        |               | 9-assoc       | 5.9e+2        | 4.0e+1        | 3.9e+1        | 5.3e-1        |
| 18-simp      | 4.6e+0        | 2.0e+0        | 1.1e-2        |               | 10-assoc      | 2.8e+4        | 5.0e+2        | 8.3e+2        | 9.6e+0        |
| 19-simp      | 1.0e+1        | 4.6e+0        | 2.2e-2        |               | 11-assoc      | MOF           | 8.2e+3        | MOF           | 2.0e+2        |
| 20-simp      | 2.2e+1        | 1.0e+1        | 4.5e-2        |               | 5-perm        | 0.0e+0        | 2.0e+0        | 6.1e-2        | 9.2e-4        |
| 21-simp      | 5.0e+1        | 2.3e+1        | 9.0e-2        |               | 6-perm        | 7.8e+0        | 2.4e+2        | 2.6e+0        | 4.2e-2        |
| 22-simp      | 1.1e+2        | 5.2e+1        | 1.8e-1        |               | 6-perm        | 2.2e+3        | 4.2e+4        | 4.3e+2        | 6.0e+0        |
| 23-simp      | 2.5e+2        | 1.1e+2        | 3.7e-1        |               | Hirsch-ex     | MOF           | MOF           | MOF           | 4.5e+2        |
| 24-simp      | 5.5e+2        | 2.5e+2        | 7.4e-1        |               | 9-cross       | 4.5e-1        | 3.7e-1        | 1.2e-1        | 1.1e-3        |
| 25-simp      | MOF           | 5.5e+2        | 1.5e+0        |               | 10-cross      | 2.8e+0        | 1.9e+0        | 4.8e-1        | 4.2e-3        |
| cyc(10,20)   | 3.3e+1        | 3.4e+1        | 2.2e+0        | 4.1e-2        | 11-cross      | 1.9e+1        | 1.0e+1        | 1.9e+0        | 1.8e-2        |
| cyc(10,21)   | 6.5e+1        | 6.7e+1        | 3.6e+0        | 8.0e-2        | 12-cross      | 1.2e+2        | 5.7e+1        | 8.2e+0        | 9.1e-2        |
| cyc(10,22)   | 1.3e+2        | 1.3e+2        | 6.2e+0        | 1.6e-1        | 13-cross      | 8.9e+2        | 3.4e+2        | 3.7e+1        | 5.1e-1        |
| cyc(10,23)   | 2.6e+2        | 2.6e+2        | 1.0e+1        | 2.9e-1        | 14-cross      | 9.6e+3        | 2.5e+3        | 1.9e+2        | 3.0e+0        |
| cyc(10,24)   | 5.9e+2        | 5.6e+2        | 1.8e+1        | 5.3e-1        | 15-cross      | MOF           | MOF           | MOF           | 1.8e+1        |
| cyc(10,25)   | 1.3e+3        | 1.3e+3        | 3.0e+1        | 9.4e-1        | 4-Birkhoff    | 6.0e-2        | 9.0e-2        | 3.7e-2        | 3.4           |
| cyc(10,26)   | 2.6e+3        | 2.5e+3        | 5.1e+1        | 1.6e+0        | 5-Birkhoff    | 2.3e+2        | 2.7e+2        | 9.9e+1        | 3.8           |
| cyc(10,27)   | 5.0e+3        | 4.7e+3        | 8.8e+1        | 2.7e+0        | 6-Birkhoff    | MOF           | MOF           | MOF           | 2.7           |
| cyc(10,28)   | 9.1e+3        | 8.3e+3        | 1.4e+2        | 4.5e+0        | 3-assoc*dual  | 4.0e-2        | 1.2e-2        | 1.3e-4        |               |
| cyc(10,29)   | 1.6e+4        | 1.4e+4        | 2.3e+2        | 7.3e+0        | 4-assoc*dual  | 2.9e+0        | 7.5e-1        | 2.5e+3        |               |
| cyc(10,30)   | 2.6e+4        | 2.3e+4        | 3.6e+2        | 1.2e+1        | 5-assoc*dual  | 5.4e+2        | 6.1e+1        | 1.5e-1        |               |
| cyc(10,31)   | 4.2e+4        | 5.7e+2        | 1.9e+1        |               | 6-assoc*dual  | 1.9e+5        | 5.3e+3        | 1.5e+1        |               |
| cyc(10,32)   | 6.6e+4        | MOF           | 2.9e+1        |               | 7-assoc*dual  |                     |               |               |               |
| cyc(10,33)   | 1.0e+5        |               | 4.4e+1        |               | 4-cross*dual  | 1.3e-1        | 1.3e-1        | 4.5e-4        |               |
| cyc(10,34)   | 1.5e+5        |               | 6.6e+1        |               | 5-cross*dual  | 3.4e+0        | 3.4e+0        | 5.3e-3        |               |
| cyc(10,35)   | 2.2e+5        |               | 9.8e+1        |               | 6-cross*dual  | 1.2e+2        | 1.2e+2        | 7.5e-2        |               |
| cyc(20,21)   | 2.1e+1        | 1.0e-2        | 4.5e-5        |               | 7-cross*dual  | 4.8e+3        | 4.7e+3        | 1.1e+0        |               |
| cyc(20,22)   | 4.8e+1        | 3.0e+2        | 9.1e-4        |               | 8-cross*dual  | 1.0e+6        | 1.0e+6        | 1.8e+1        |               |
| cyc(20,23)   | 2.6e+2        | MOF           | 2.1e-3        |               |               |               |               |               |               |
| cyc(20,24)   | MOF           |               | 5.2e-3        |               |               |               |               |               |               |

Institut für Mathematik, Freie Universität Berlin, Germany
E-mail address: jonathan.kliem@fu-berlin.de

Fakultät für Mathematik, Ruhr-Universität Bochum, Germany
E-mail address: christian.stump@rub.de