Symmetric lexicographic subset reverse search for the enumeration of circuits, cocircuits, and triangulations up to symmetry

Draft

Jörg Rambau

September 20, 2022

Abstract

This paper introduces three variants of the enumeration algorithm symmetric lexicographic subset reverse search and finds specializations for three applications: cocircuits, circuits, and triangulations of point configurations. There are two new methods presented to check the lexicographic minimality of a subset in its orbit: the critical-element method and the modified switch-table method. Moreover, new application-dependent methods to reduce the number of necessary enumeration nodes are introduced: rank-pruning for cocircuits and lex-pruning for triangulations. For circuits, a compact data structure, the column representation matrix, is introduced that allows the detection of signed circuits only by admissible column operations. With an implementation of the ideas in the software package TOPCOM, in all three applications known benchmarks can be computed faster by a large margin, and new numbers, like the number of cocircuits of the 9-cube, the number of circuits of the 8-cube, and the number of all triangulations of the product of a 5- and a 3-simplex can be computed for the first time.

Contents

List of Algorithms

1 Introduction

This paper systematically studies how to enumerate subsets of a finite set up to symmetry based on a specializtion of the reverse-search paradigm to subsets orbits. In order to demonstrate the potential of this approach, three applications are presented: for a point or vector configuration, enumerate up to symmetry all its cocircuits, all its circuits, and all its triangulations, respectively. In all three applications the scales of problem instances that can be handled are extended significantly.

Let us start with a review of related literature. A fairly general account for the complexity of enumeration algorithms can be found in [11], where the objects to be enumerated are considered as the results of certain abstract closure operations. All the enumeration problems there do not consider symmetries.

An example where the problem of ignoring symmetries is apparent, is the enumeration of circuits of a matroid. This brings us to the second application in this paper, which concerns *signed circuits* of a point or vector configuration. There is an incremental-polynomial-time algorithm for the enumeration of circuits in [9] based on the exchange axiom for circuits, which improves on older algorithms like the one in [12] based on using bases for constructing circuits. However, it is not clear how symmetries could be exploited in this algorithm. Moreover, the nature of the exchange axiom shows that the time to add one more circuit is at least quadratic in the number of already computed circuits, which seems prohibitive at least for the larger examples in this paper.

Dual to this is the enumeration of *signed cocircuits* of a point or vector configuration, pointing at the first application in this paper. In [1] all hyperplanes spanned by vertices of the d-dimensional hypercube C^d were classified and enumerated up to $d = 8$ (12 days of cpu time on a computer that was fast according to the standards of that time). The authors estimated a cpu time of 35 years for $d = 9$ for their method based on the classification of normal vectors. There seems to be no documented algorithm or code enumerating the hyperplanes in a general point or vector configuration.

The top-level enumeration method in this paper can be seen as a specialization of *reverse search* [2] to subsets up to symmetry.

Reverse search has been specialized to the enumeration of orbits in [6] and parallelized in [3]. The ideas therein were then the basis for parallel enumeration up to symmetry of all *subregular triangulations*, a certain subset of the component of regular triangulations, in [7]. This now directly shifts the focus to the third application in this paper. The freely available package `mptopcom` significantly extended the scale of instances that could be handled compared to the earlier `TOPCOM` [14], which itself was a large step forward from de Loera's pioneering `maple`-code `PUNTOS` from his thesis [4]. The code `mptopcom` was later specialized for cyclic polytopes to generate some new numbers [8] extending the computational results in [14, 15].

All these attempts are *flip-based*, i.e., they explore the flip graph of triangulations, where two triangulations share an edge if they are connected by a *bistellar flip*. This is a generalization of swapping diagonals in a convex quadrilateral in dimension two (see [5] for definition in all dimensions). Since Santos's triangulation without flips in [17] it is known that flip-based algorithms may not find all triangulations in general. TOPCOM [14] was the first available software to enumerate all triangulations by building them simplex-by-simplex, which will be called an *extension-based* algorithm. However, the examples that could be computed were only toy-size examples. In [6] the extension-based enumeration of triangulations was reduced to the enumeration of maximal cliques in the *proper-intersection graph* of all simplices. However, no computational results were given, and the results in this paper show evidence for the fact that a pure max-clique enumeration does not work in practice, since there are too many maximal cliques that do not correspond to a triangulation but to a partial triangulation that cannot be extended. The enumeration of all triangulations corresponds to an enumeration of all vertices of the *universal polytope* [10]; since no full outer description of this polytope is available, the vertex enumeration problem for it is non-trivial.

Besides the fact that extension-based algorithms reach all triangulations, there is at least one other motivation for them: If the search shall be restricted to triangulations using only special simplices (like empty simplices or unimodular simplices), then the flip-based algorithms have to explore the whole flip-graph and filter ex-post by the wanted triangulations, whereas an extension-based algorithm can exclude the unwanted simplices from consideration right a-priori.

The specialization of reverse-search to feasible subsets of a finite set is probably folklore, but a specialization exploiting the lexicographic ordering of subsets to enumerate subset *orbits* was first formalized (in a different language) in [13].

Let us turn to the contributions of this paper. The top-level method used in this paper consists of a generic enumeration framework for the enumeration of all orbits of a *downset*, i.e., a set of subsets closed under taking subsets. This framework is called *Symmetric Lexicographic Subset Reverse Search* (`SymLexSubsetRS`) in this paper. This paper studies three variants of `SymLexSubsetRS`: enumerate orbits of *maximal* elements *in* a downset, enumerate orbits of *minimal* elements *not in* a downset, and enumerate orbits of *feasible* antichains *in* a downset. The generic algorithm `SymLexSubsetRS` and the variant for feasible subsets was essentially presented already in [13], and the method in this paper can be seen as variants, refinements, and new specializations of it and its subroutines. All applications have been implemented in the TOPCOM package, which is freely available under the Gnu Public Licence on the author's webpage.

The original contributions of this paper are twofold: for the general framework, two alternative checks of lexicographic minimality of a subset in its orbit (called the *lex-min check*) are proposed; for the particular applications, new methods are presented for recognizing that a subset cannot be extended to a feasible subset by adding larger elements (called the *lex-ext check*).

Concerning the *lex-min check*, the first new alternative is the *modified switch-table method*. It combines the ideas in [13] with the switch-table method in [7]. This alternative works best for symmetry groups whose order is large compared to the degree. The enumeration of circuits and cocircuits is an appropriate use-case.

The second alternative is the *critical-element method*. It is based on some new theory presented in Section 4. This alternative is mainly interesting for cases in which the symmetry group is of moderate order, i.e., is given as a list of all permutations in it, and of a degree in the same order of magnitude, i.e., there are at least as many elements as there are permutations. The enumeration of all triangulations ususally fits into this scheme.

The advantage of both alternatives compared to [13] is that they completely avoid the generation of additional stabilizer groups inside tight loops. This turned out to be advantageous in the applications of this paper.

Concerning the *lex-ext checks* for the applications, the new rank-based *rank-pruning* accelerates the enumeration of cocircuits up to symmetry. This allowed the first ever enumeration of all hyperplanes in the 9-cube C^9 up to symmetry in less than 14 hours. Although it is difficult to tell how fast the code from [1] would run on today's computers, TOPCOM's enumeration algorithm works for general configurations and does not use any theory about cubes like the algorithm in [1].

For the enumeration of circuits no effective lex-ext check was found so far. Still, the algorithm could compute some new numbers, among them the numbers of circuits up to symmetry of the hypercubes C^6, C^7, and C^8. Note that in order to enumerate circuits one can also enumerate cocircuits in the *Gale-transform* (see [5] for more background on this). Whether or not this is faster or slower usually depends on the rank and the corank of the configuration. Having specialized algorithms for both means that one can pick the respective faster strategy.

For the enumeration of triangulations up to symmetry, the new lex-ext check *lex-pruning* is the single most important progress. It is based on the property of any triangulation that each interior facet of a simplex is covered by another simplex [5, Cor. 4.1.32]. From this one can derive the rather tight lex-ext check *full-pruning*. The new lex-ext check lex-pruning heavily exploits on the lexicographic ordering of all simplices and their interior facets in all data structures involved. While full-pruning has to check many subset relations, lex-pruning only compares two certain integers. Still it is almost as effective as full-pruning.

The paper is organized as follows. Section 2 reviews some important notions and algorithms. Moreover, the the notational conventions are fixed. Section 3 is a short general problem statement for what the paper tries to achieve. In Section 4 the theoretical considerations for the new lex-min checks are proven. Section 5 is devoted to the variants of the top-level algorithm and the lex-min checks that are relevant in general. The discussion of the three application starts in Section 6 with some common preliminaries on point and vector configurations. Then, Sections 8 through 10 present the new results that are relevant for each application individually. Finally, Section 11 contains a summary and some conclusions.

2 Preliminaries

In this section, some basic notions and notation are introduced in the form we will use it. Moreover, some known algorithms are formulated in our framework.

Let $[n]$ denote the set of integers $\{1,2,\dots,n\}$. For $k \in \mathbb{Z}$ the set of all k-element subsets of $[n]$ is written as $\binom{[n]}{k}$. The power set of $[n]$ is denoted by $2^{[n]}$.

The elements of $\binom{[n]}{k}$ are totally ordered by the *(subset-)lexicographic order* given by $\{s_1,\dots,s_k\} <_{\text{lex}} \{r_1,\dots,r_k\}$ if there is a $j \in [k]$ with $s_j < r_j$ and $s_i = r_i$ for all $1 \le i < j$. This total order can be extended to $2^{[n]}$, but this will not be used.

For an undirected graph $G = (V,E)$ and a node of it $v \in V$ the set $N(v)$ is the set of nodes connected to v by an edge in E. Its cardinality $|N(v)|$ is the degree $d(v)$ of $v \in V$, and the maximum of all degrees over all nodes is denoted by $d^{\max}(G)$ (or $d^{\max}$ if G is clear from the context).

The symmetric group on n elements is considered as the set of bijections from $[n]$ to itself. It is denoted by $\mathfrak{S}_n$, and subgroups of it are usually called $\mathfrak{G}$. For a subgroup $\mathfrak{G}$ of $\mathfrak{S}_n$ and a finite set Ω, a map $\phi : \mathfrak{G} \times \Omega \to \Omega$ is a (left) group action of $\mathfrak{G}$ on Ω if $\phi(\pi \cdot \sigma, \omega) = \phi\big(\pi, \phi(\sigma,\omega)\big)$ for all $\pi, \sigma \in \mathfrak{G}$ and all $\omega \in \Omega$. Most of the times, certain clearly induced group actions ϕ are denoted by $\pi(\omega) := \phi(\pi,\omega)$ for $\pi \in \mathfrak{G}$ and $\omega \in \Omega$, leading to $(\pi \cdot \sigma)(\omega) = \pi\big(\sigma(\omega)\big)$. Given such a group action, the *stabilizer subgroup* of $\omega \in \Omega$ in $\mathfrak{G}$ is $\mathfrak{G}_\omega := \{\pi \in \mathfrak{G} : \pi(\omega) = \omega\}$. The $\mathfrak{G}$*-orbit* of ω is $\mathfrak{G}(\omega) := \{\pi(\omega) : \pi \in \mathfrak{G}\}$.

More specifically: For a permutation $\pi \in \mathfrak{S}_n$ and a graph G with node set $V = [n]$ and edge set E let $\pi\big(\{v,w\}\big) = \big\{\pi(v), \pi(w)\big\}$ denote the induced action of π on edges, for all edges $\{v,w\} \in E$. The definition $\pi(E) := \{\pi(e) : e \in E\}$ induces a new graph $\pi(G) = \big([n], \pi(E)\big)$ on the same node set. This defines an action of $\mathfrak{S}_n$ on the set of all graphs on the node set $[n]$. The automorphism group $\operatorname{Aut}(G)$ of a graph $G = ([n],E)$ is the set of all $\pi \in \mathfrak{S}_n$ with $\pi(G) = G$. The elements of $\operatorname{Aut}(G)$ are the *symmetries of* G.

Similarly, for $\pi \in \mathfrak{S}_n$ the induced action of π on any k-element subset $S = \{s_1,\dots,s_k\} \in \binom{[n]}{k}$ is denoted by $\pi(S) = \big\{\pi(s_1),\dots,\pi(s_k)\big\} \in \binom{[n]}{k}$. For a subset $\mathscr{S} = \{S_1,\dots,S_m\}$ of 2^n the induced action of π on $\mathscr{S}$ is denoted by $\pi(\mathscr{S}) = \big\{\pi(S_1),\dots,\pi(S_m)\big\}$. The automorphism group $\operatorname{Aut}(\mathscr{S})$ is the set of all $\pi \in \mathfrak{S}_n$ with $\pi(\mathscr{S}) = \mathscr{S}$.

The analogous concept can be used for sets of subsets of $[n]$. For example, in one application there is the induced action of a permutation $\pi \in \mathfrak{S}_n$ on a triangulation $\mathscr{T}$ of a point configuration with n labeled points given by the label sets of its maximal simplices. That action is denoted by $\pi(\mathscr{T})$ as well.

In the remainder of this section, the general-purpose algorithm Symmetric Lexicographic Subset Reverse Search (`SymLexSubsetRS`) is recapitulated in the language of the reverse-search paradigm from [2]. Starting at the basic Reverse Search, the extensions to orbits and the specializations to subsets are introduced one-by-one. The algorithms in this section are not new, but a summary of all variants in a unified notational environment is helpful to understand the extensions.

As a simplification, a recursive form is used for presentation, which usually increases the memory consumption of the algorithms. However, the more memory-efficient original Reverse-Search framework (with non-recursive backtracking by pivoting) can be applied to all presented algorithms. In this paper, recursive implementations have been used throughout, since they were faster in the presented applications.

First, the original Reverse Search is formulated [2]. Reverse Search (RS) is an algorithm to enumerate the nodes of a graph $G = (V,E)$ with known maximal degree $d^{\max}$. The graph is implicitly given by an adjacent-nodes function $\operatorname{Adj} : V \times [d^{\max}] \to V \cup \{\mathsf{NULL}\}$ that returns for each node $v \in V$ and each integer $i \in [d^{\max}]$ the ith neighbor of v if $i \le d(v)$ and NULL if $d(v) < i \le d^{\max}$. A recursive representation of Reverse Search in pseudo-code is shown in Algorithm 1.

The run-time complexity of RS is $O(d^{\max}\tau(\operatorname{Adj})|V| + \tau(p)|E|)$, where $\tau(f)$ denotes the maximal time to compute a function value of a function f. Since $2|E| \le d^{\max}|V|$, this run-time complexity is in particular in $O(d^{\max}(\tau(\operatorname{Adj}) + \tau(p))|V|)$. This in particularly interesting in the case where $\tau(\operatorname{Adj})$ and $\tau(p)$ do not depend on $|V|$. In that case, RS is linear in the output size [2].

If one has a group $\mathfrak{G}$ acting on G one is usually only interested V up to symmetry. In other words,

```
Algorithm: RS(Adj, $\phi$, $p$, $v$)
Input: a connected graph $G = (V, E)$ with maximal degree $d^{\max}$, implicitly given by an adjacent-nodes
       function $\mathrm{Adj} : V \times [d^{\max}] \to V \cup \{\mathsf{NULL}\}$, a cost function $\phi : V \to \mathbb{R}$ with $\phi(v) \neq \phi(w)$ for all
       $v \neq w$ in $V$ and $\phi(v^*) = \min_{v \in V} \phi(v)$, a pivot function $p : V \to N(V) \cup \{\mathsf{NULL}\}$ with
       $\phi\big(p(v)\big) < \phi(v)$ for all $v \in V \setminus \{v^*\}$ and $p(v^*) = \mathsf{NULL}$, a seed node $v$ in $V$
Output: the number of nodes in $V$
/* build a depth-first-search tree with root node $v$:                        */
$c \leftarrow 1$;                                                          /* count $v$ */
for $j = 1, \dots, d^{\max}$ do                                  /* iterate over neighbors of $v$ */
    $w \leftarrow \mathrm{Adj}(v, j)$;                                       /* get $j$th neighbor */
    if $w = \mathsf{NULL}$ then                              /* if we are past the last neighbor */
        break;                                                        /* exit the loop */
    if $v = p(w)$ then                                              /* if $w$ pivots to $v$ */
        $c \leftarrow c + $RS(Adj, $\phi$, $p$, $w$) ;                                       /* recurse */
return c;
```

Algorithm 1: The generic reverse search algorithm (cf. [2]); it enumerates the number of nodes with node-cost function worse than v in a graph that is implicitly given by the adjacent-nodes function $\mathrm{Adj}(v, j)$ for all nodes $v \in V$; with $v = v^*$ the algorithm enumerates V

the number of $\mathfrak{G}$-orbits of V is asked for. How the reverse search principle can be adapted to this setting, was first presented in [6]. The idea is to extend the adjacent-nodes function Adj to $\mathfrak{G}$-orbits in the canonical way and to modify the pivot-function to consist of two steps: In the first step, the ϕ-minimal element in the current orbit is taken; in the second step, the ϕ-reducing pivot-function on G is followed. Thus, another orbit is reached in case the minimal element of the current orbit was no global minimum already. Algorithm 2 shows a detailed pseudo-code representation.

A specialized form of Reverse Search arises when a set of "feasible" subsets of a finite shall be enumerated by adding elements, checking feasibility one-by-one, and back-tracking. This automatically will touch many subsets of feasible sets. Thus, one can restrict to the enumerations of downsets, i.e., sets of subsets, where each subset of a feasible set is feasible itself. Algorithm 3 shows the specialization of Reverse Search, which is a folklore observation.

A notable further specialization can be seen in Algorithm 4: by using the natural order for extending subsets by a new element one can guarantee that the subsets are found in lexicographic order.

If downsets shall be enumerated up to symmetry, this enumeration in lexicographic order simplifies affairs substantially: Since all subsets appear in lexicographic order, we know that in each orbit the lex-min element is found first. Thus, if we take the lex-min element in each orbit as the canonical representative, there is no need for computing canonical representatives for all found elements anymore. The only thing needed is to check whether or not the found element can be lex-decreased *at all* by the action of an element in $\mathfrak{G}$. If so, the found element is not canonical, and since canonical elements are found first, its orbit has been counted already. An important extra-feature is that lex-leading subsets of a subset that is lex-minimal in its orbit are lex-minimal in their orbits as well (see Lemma 2 for a formal proof). The resulting algorithm (Algorithm 5) was proposed in [13] together with suggestions how the lex-min check can be implemented.

With this, the enumeration of subsets up to symmetry is reduced to checking, whether a subset is lex-min in its orbit, and to checking, whether a subset is feasible. In particular in cases, where the downset to be enumerated up to symmetry has only been implicitly defined as a downset by a set of really interesting subsets together with all its subsets, one needs to answer the question, whether or

```
Algorithm: SymRS(Adj, $\phi$, $p$, $\mathfrak{G}$, $v$)
Input: a connected graph $G = (V, E)$ with maximal degree $d^{\max}$, implicitly given by an adjacent-nodes
        function Adj : $V \times [d^{\max}] \to V \cup \{\mathsf{NULL}\}$, a cost function $\phi : V \to \mathbb{R}$ with $\phi(v) \neq \phi(w)$ for all
        $v \neq w$ in $V$ and $\phi(v^*) = \min_{v \in V} \phi(v)$ , a pivot-function $p : V \to N(V) \cup \{\mathsf{NULL}\}$ with
        $\phi\big(p(v)\big) < \phi(v)$ for all $v \in V \setminus \{v^*\}$ and $p(v^*) = \mathsf{NULL}$, a subgroup $\mathfrak{G}$ of Aut($G$) and a
        canonical-representative function Canonical($v$) =: $\check{v}$ with $\phi(\check{v}) < \phi(w)$ for all $w \in \mathfrak{G}(v)$, a
        seed node $\check{v}$ in $V$ that is the canonical representative of $\mathfrak{G}(\check{v})$
Output: the number of $\mathfrak{G}$-orbits in $V$
/* build a depth-first-search tree with root node $\check{v}$:                              */
$c \leftarrow 1$ ;                                                                   /* count $\check{v}$ */
$\check{W} \leftarrow \{\check{v}\}$ ;                              /* collects already processed canonicals */
for $j = 1, \ldots, d^{\max}$ do                                      /* iterate over neighbors of $\check{v}$ */
    $w \leftarrow$ Adj($\check{v}, j$) ;                                                 /* get $j$th neighbor */
    if $w = \mathsf{NULL}$ then                                      /* if we are past the last neighbor */
        break ;                                                                      /* exit the loop */
    $\check{w} \leftarrow$ Canonical($w$) ;                             /* compute canonical representative */
    if $\check{w} \notin \check{W}$ then                                                       /* if $\check{w}$ is new */
        $u \leftarrow p(\check{w})$ ;                                                      /* compute pivot */
        $\check{u} \leftarrow$ Canonical($u$) ;                         /* compute canonical representative */
        if $\check{u} = \check{v}$ then                                       /* if $\mathfrak{G}(w)$ pivots to $\mathfrak{G}(\check{v})$ */
            $\check{W} \leftarrow \check{W} \cup \{\check{w}\}$ ;                  /* add $\check{w}$ to set of processed canonicals */
            $c \leftarrow c +$ RS(Adj, $\phi$, $p$, $\mathfrak{G}$, $\check{w}$) ;                                   /* recurse */
return c;
```

Algorithm 2: The standard application of reverse-search to the graph of all orbits $\mathfrak{G}(v)$, where the canonical representative of each orbit is its ϕ-minimal element (cf. [6]); two orbits are connected by an edge whenever at least one (and, thus, each) representative of one orbit is connected to a representative of the other orbit by an edge in E; the node-cost function in the orbit graph is given by the node-cost function on the orbits' canonical representatives; the pivot function in the orbit graph is given by the pivots in G of the canonical representatives; the algorithm enumerates all orbits with worse node costs than the orbit of the seed node v; with $v = v^* = \check{v}^*$ it enumerates V up to symmetries in $\mathfrak{G}$

```
Algorithm: SubsetRS($n, \mathscr{D}, S$)
Input: $n \in \mathbb{N}$, a downset $\mathscr{D}$ of $2^{[n]}$, a seed set $S \in \mathscr{D}$
Output: the number of elements $S'$ in $\mathscr{D}$ with $S \subseteq S'$
/* build a depth-first-search tree with root node $S$:                      */
$c \leftarrow 1$;                                                  /* count $S$ */
for $j \in [n] \setminus S$ with $j > \max(S)$ do      /* for extensions with new max */
    $S' \leftarrow S \cup \{j\}$;                                /* build new set */
    if IsInDownset($S', \mathscr{D}$) then          /* if new set belongs to $\mathscr{D}$ */
        $c \leftarrow c +$ LexSubsetRS($n, \mathscr{D}, S'$);            /* recurse */
return c;
```

Algorithm 3: The standard application of reverse-search to the Hasse-diagram of a downset $\mathscr{D}$ of subsets of an n-element set, partially ordered by inclusion; it uses for $\phi(S)$ the position of S in the lexicographic order of $2^{[n]}$ and $p(S) = S \setminus \{\max(S)\}$; it enumerates all subsets containing S in a downset $\mathscr{D}$ of the power set $2^{[n]}$ of $[n]$; with $S = \emptyset$ the algorithm enumerates $\mathscr{D}$; the performance greatly depends on the implementation of the $\mathscr{D}$-membership test IsInDownset

```
Algorithm: LexSubsetRS($n, \mathscr{D}, S$)
Input: $n \in \mathbb{N}$, a downset $\mathscr{D}$ of $2^{[n]}$, a seed set $S \in \mathscr{D}$
Output: the number of elements $S'$ in $\mathscr{D}$ with $S \subseteq S'$
/* build a depth-first-search tree with root node $S$:                      */
$c \leftarrow 1$;                                                  /* count $S$ */
for $j = \max(S)+1, \dots, n$ do      /* ordered traversal of new maximal elements */
    $S' \leftarrow S \cup \{j\}$;                                /* build new set */
    if IsInDownset($S', \mathscr{D}$) then          /* if new set belongs to $\mathscr{D}$ */
        $c \leftarrow c +$ LexSubsetRS($n, \mathscr{D}, S'$);            /* recurse */
return c;
```

Algorithm 4: An specialized implementation of SubsetRS($n, \mathscr{D}, S$) that exploits the straightforward set-valued inverse of the pivot function and traverses the pivot-inverse according to the total order in subsets of $[n]$; the result is that the algorithm does not need to scan all supersets of a subset; moreover, it automatically enumerates all subsets in $\mathscr{D}$ containing S in lexicographic order, and for $S = \emptyset$ it enumerates $\mathscr{D}$; note that since all finite sets are in one-to-one correspondence to some $[n]$ by an arbitrary indexing, this specialization can be used to enumerate all downsets of subsets of any finite set; the performance greatly depends on the implementation of the $\mathscr{D}$-membership test IsInDownset

```
Algorithm: SymLexSubsetRS($n, \mathscr{D}, \mathfrak{G}, S$)
Input: $n \in \mathbb{N}$, a downset $\mathscr{D}$ of $2^{[n]}$, a subgroup $\mathfrak{G}$ of the automorphism group of $\mathscr{D}$, a seed set $S \in \mathscr{D}$
Output: the number of $\mathfrak{G}$-orbits in $\mathscr{D}$ with $S \subseteq S'$
/* build a depth-first-search tree with root node $S$:                        */
$c \leftarrow 1$;                                                      /* count $S$ */
for $i = \max(S)+1, \ldots, n$ do          /* ordered traversal of new maximal elements */
    $S' \leftarrow S \cup \{i\}$;                        /* add a new element on the right */
    if IsLexMin($S', \mathfrak{G}$) then              /* if new set is lex-min in its orbit */
        if IsInDownset($S', \mathscr{D}$) then               /* if new set belongs to $\mathscr{D}$ */
            $c \leftarrow c +$ SymLexSubsetRS($n, \mathscr{D}, \mathfrak{G}, S'$);           /* recurse */
return c;
```

Algorithm 5: A specialized implementation of the combination of LexSubsetRS and SymRS for orbits of subsets in a downset $\mathscr{D}$ of $2^{[n]}$ (cf. [13]); since LexSubsetRS enumerates all subsets in lexicographic order, in each orbit the lexicographically-minimal, i.e., the canonical subset is found first, and therefore a lex-min check can replace the computation of the canonical subsets; therefore, it automatically enumerates all lex-min representatives of orbits of subsets in $\mathscr{D}$ containing S in lexicographic order, and for $S = \emptyset$ it enumerates $\mathscr{D}$; note that since all finite sets are in one-to-one correspondence to some $[n]$ by an arbitrary indexing, this specialization can be used to enumerate all orbits of downsets of subsets of any finite set; the performance greatly depends on the implementation of the lex-min check IsLexMin and the $\mathscr{D}$-membership test IsInDownset

not a subset is a subset of a really interesting set. This problem arises in all of the applications in this paper.

The methods proposed in the literature to check whether a subset is lex-min in its orbit all aim at different use-cases. Some are used in the case of small groups, for which all elements can be easily precomputed and traversed. For such groups, a taylor-made new method is developed in Section 4. This turned out to be the fastest method for the enumeration of triangulations, where the typical orders of automorphism groups are in the thousands.

The lex-min check for groups with a larger order (like in the millions and above), there are a *recursive method* (for subsets only) proposed in [13] and a *switch-table*-method (originally for vectors, which is more general) from [7]. A new combination of the two (for subsets) turned out to be the fastest method for the enumeration of cocircuits and circuits in our test cases. Thus, switch tables are briefly introduce next.

Given a subgroup $\mathfrak{G}$ of $\mathfrak{S}_n$ a *switch table* is a function $\mathrm{st}(\cdot,\cdot) : [n] \times [n] \to \mathfrak{G}$ with either $\mathrm{st}(i, j) = \pi$ so that $\pi(j) = i$ and $\pi(k) = k$ for all $k < i$, if such a $\pi \in \mathfrak{G}$ exists, and $\mathrm{st}(i, j) = \mathrm{id}$ otherwise. A switch table need not be unique. The entries $\mathrm{st}(i, j)$ are called *switches*. An entry of a switch table is *trivial* if it is the identity. A row of a switch table is *effective* if it contains at least one non-trivial switch. The set effRowSet of all row indices i so that $\mathrm{st}(i,\cdot)$ is non-trivial is called the *effective row set* of $\mathrm{st}(\cdot,\cdot)$. The (non-empty) set of column indices of non-trivial switches in an effective row $i \in$ effRowSet is the *effective column set* of i, denoted by effColSet(i).

One key property of a switch table is that each element of $\mathfrak{G}$ can be written as an essentially unique product of switches consisting of at most one switch from each row. The key in the method to find a lex-smaller element in the orbit of a subset is that not all switches can contribute to a lex-decreasing switch-product, so that not all products need to be considered. This way, the check actually traverses significantly (depending on the group action) fewer elements (= suitable switch products) than $\mathfrak{G}$ has elements. A straight-forward version of the algorithm for subsets is presented in

Algorithm 6. Details for vectors can be found in [7]. The algorithm uses global auxiliary data, which is given by a switch table $\mathsf{ST} = \mathrm{st}(\cdot,\cdot)$ for $\mathfrak{G}$.

```
Algorithm: IsLexMin_viaSwitches($i, S, S', \mathfrak{G}, \mathsf{ST}$)
Input: an integer $i \in [1,n]$, a subset $S$ (the original subset), a subset $S'$ (the subset mapped by a partial
       switch product), a subgroup $\mathfrak{G}$ of $\mathfrak{S}_n$, and a switch table $\mathsf{ST}$ of $\mathfrak{G}$
Output: TRUE if $S = \text{lex-min}\,\mathfrak{G}(S)$ and FALSE otherwise
/* check for empty set:                                                    */
if $S = \emptyset$ then
    return TRUE;
/* are we beyond the effective row set?                                    */
if $i > \max \mathrm{effRowSet}(\mathsf{ST})$ then
    return $(S' \not<_{\mathrm{lex}} S)$;
/* recursively check a switch product with leading identity:               */
if IsLexMin_viaSwitches($i+1, S, S', \mathfrak{G}, \mathsf{ST}$) = FALSE then
    return $(S' \not<_{\mathrm{lex}} S)$;
$\mathsf{GS} \leftarrow \{ j \in \mathrm{effColSet}(k) \mid i \notin S', j \in S' \}$;      /* switches lex-decreasing $S'$ */
if $\mathsf{GS} = \emptyset$ then
    $\mathsf{GS} \leftarrow \{ j \in \mathrm{effColSet}(k) \mid |S' \cap \{i,j\}| \in \{0,2\} \}$;      /* neutral switches */
for $j \in \mathsf{GS}$ do
    $S'' \leftarrow \mathsf{ST}[i][j](S')$;                                       /* map subset */
    if $S'' <_{lex} S$ then
        return FALSE;                               /* decreasing switch product found */
    /* recurse with mapped subset:                                         */
    if IsLexMin_viaSwitches($i+1, S, S'', \mathfrak{G}, \mathsf{ST}$) = FALSE then
        return FALSE;
return TRUE;                                     /* no decreasing switch product found */
```

Algorithm 6: The switch-table method from [7] specialized to check whether a subset S' cannot be mapped to a lex-smaller subset than S by the permutations in $\mathfrak{G}$ stabilizing the elements $\{1,\ldots,i-1\}$; for $i=0$ and $S' = S$, it checks whether S is lex-min in its $\mathfrak{G}$-orbit; its global data is a switch table for $\mathfrak{G}$

3 Problem Statement

In this paper, the following questions concerning the algorithm SymLexSubsetRS and some variants are studied in detail:

- How can the subroutine IsLexMin of SymLexSubsetRS be implemented efficiently *in general*?
- How can the subroutines of SymLexSubsetRS and variants be implemented efficiently *for each of our applications.* More specifically:
  - How can one effectively prune subsets of vectors that are not lex-initial segments of a circuit/cocircuit?
  - How can one effectively prune subsets of simplices that are not lex-initial segments of a triangulation of the point configuration?

The term *efficiently* here is not meant in any complexity-theoretic meaning. For the algorithms in this paper no polynomial general bound could be derived on the complexity in terms of input and output size so far.

4 Theoretical Foundations

For some of the upcoming arguments the following characterization of the lexicographic order on k-element subsets of $[n]$ is used.

Lemma 1 (Lexicographic order on k-subsets)**.** *Let $n \in \mathbb{N}$. Moreover, let S and R be k-element subsets of $[n]$. Then S is lexicographically smaller than R if the minimal element of their symmetric difference is in S. In formulae:*

$$S <_{lex} R \iff \min(S \bigtriangleup R) \in S, \tag{1}$$

where $\min \emptyset = \infty$.

The following lemma states that if subsets are built element-by-element with backtracking and only the lexicographically minimal ones in their orbits are followed, then one will reach all subsets that are lexicographically minimal in their orbits. This result was already presented in [13]; here, a slightly extended proof is shown.

Lemma 2 (cf. [13])**.** *Let S be a subset of $[n]$ and $S^- := S \setminus \{\max S\}$. Then, for all subgroups $\mathfrak{G}$ of $\mathfrak{S}_n$ we have:*

$$S = \text{lex-min}\, \mathfrak{G}(S) \Rightarrow S^- = \text{lex-min}\, \mathfrak{G}(S^-) \tag{2}$$

Proof. Let S be lex-min in its $\mathfrak{G}$-orbit. Assume, for the sake of contradiction, that S^- is not lex-min in its $\mathfrak{G}$-orbit. Then, there is a set R which is lex-smaller than S^- and a permutation $\pi \in \mathfrak{G}$ with $\pi(S^-) = R$. That means, $\min(S^- \bigtriangleup R) \in R$. Moreover, $\pi(\max S) \notin R$. Consider $R^+ := R \cup \{\pi(\max S)\}$. Since $\max S > \min(S^- \setminus R) > \min(R \setminus S^-) = \min(S^- \bigtriangleup R)$ we know that $\min(S \setminus R) > \min(S^- \bigtriangleup R)$. Moreover, $\min\bigl(S \setminus R^+\bigr) \geq \min(S \setminus R)$. Thus, $\min(S \bigtriangleup R^+) \in R^+$, and, hence, $R^+ = \pi(S)$ is lex-smaller than S: a contradiction to the assumption that S is lex-min in its orbit. □

The main theoretical question in the general part of this paper is: how can one find out whether or not a given k-subset is lexicographically minimal in its $\mathfrak{G}$-orbit. The known methods work along variants of strong generating sets of the symmetry group $\mathfrak{G}$. They look for words combined from certain generators that lexicographically decrease a subset. The disadvantage of such methods is that the action of a word on a subset must be evaluated many times. Each such action requires either the computation of a product of permutations and the action of the product on a subset or (better) the repeated computation of the action of a permutation on the subset at hand.

Still: If the symmetry group is very large, generator-set methods are superior. However, the applications in this paper very often concern smaller symmetry groups that can easily be enumerated first. How can one reduce the number of evaluations of actions on subsets in that check? Remember that the orbits' lexicographically-minimal subsets are built element-by-element. That is, for the check of whether or not a given k-subset S is lexicographically minimal in its orbit, one can exploit that its predecessor-subset S^- with $k-1$ elements is already known to be lexicographically minimal in its orbit. The new question is: has the addition of the new maximal element led to the existence of a permutation that lexicographically decreases the new subset S in its orbit?

The new idea in this paper is to keep the information about *why* the predecessor subset S^- is lexicographically minimal in its orbit. The reason is that for each permutation $\pi \in \mathfrak{G}$ one has

$$\min\bigl(S^- \bigtriangleup \pi(S^-)\bigr) \notin \pi(S^-). \tag{3}$$

These minimal elements of the symmetric differences of subsets and their images are critical for the question at hand, which motivates the following definition:

Definition 1. Let $n \in \mathbb{N}$ and $\mathfrak{G}$ be a subgroup of $\mathfrak{S}_n$. For a given k-subset S the *critical-element table with respect to S* is defined as follows:

$$\operatorname{critelem}_S \colon \begin{cases} \mathfrak{G} & \to & [n] \cup \{\infty\}, \\ \pi & \mapsto & \min\big(S \mathbin{\triangle} \pi(S)\big), \end{cases} \tag{4}$$

where $\min \emptyset := \infty$. The function value $\operatorname{critelem}_S(\pi)$ is called the *critical element of π with respect to S*.

From the critical-element table, some important of properties of the action of a permutation π on the given subset S can be read-off easily:

Lemma 3. *(i) The stabilizer of S in $\mathfrak{G}$ is the set of permutations with critical element ∞.*

(ii) A subset S is lexicographically minimal in its $\mathfrak{G}$-orbit if and only if $\operatorname{critelem}_S(\pi) \notin \pi(S)$ *for all* $\pi \in \mathfrak{G}$.

(iii) Let S be a non-empty subset of $[n]$ and $S^- := S \setminus \{\max S\}$. Moreover, assume that S^- is lexicographically minimal in its $\mathfrak{G}$-orbit. Then a permutation $\pi \in \mathfrak{G}$ lexicographically decreases S if and only if one of the following cases occurs:

I. $\operatorname{critelem}_{S^-}(\pi) = \infty$ *and* $\pi(\max S) < \max S$

II. $\operatorname{critelem}_{S^-}(\pi) \in S^-$ *and* $\pi(\max S) < \operatorname{critelem}_{S^-}(\pi)$ *or*

III. $\operatorname{critelem}_{S^-}(\pi) \in S^-$, $\pi(\max S) = \operatorname{critelem}_{S^-}(\pi)$ *and* $\operatorname{critelem}_S(\pi) \in \pi(S)$ $\square$

Call the application of this the *critical-element method* for checking lexicographic minimality of a subset in its orbit.

The crucial gain of this lemma is the following: given the critical-element table with respect to S^-, one can check lexicographic minimality of S in its orbit without actually computing $\pi(S)$, with the only exception when π maps the new element of S exactly to the critical element of S^-. And this exception roughly happens for a $\frac{1}{n}$-fraction of the permutations, on average.

There are now two ways to implement the critical-element method:

1. iterate over all permutations and apply Lemma 3 to each of them (the *iteration-based critical-element method*);
2. first, from certain fixed, preprocessed subsets of permutations, compute the subsets of permutations that

   (a) certainly lexicographically decrease the given subset, and if empty,

   (b) possibly lexicographically decrease the given subset.

   If the subset of certainly lexicographically decreasing permutations is non-empty, the subset is not lexicographically minimal in its orbit; if it is empty, iterate over the possibly lexicographically decreasing permutations and apply Lemma 3 to only those (the *set-based critical-element method*).

Section 5 presents in detail algorithms for the iteration-based and the set-based method to decide whether or not a subset is lex-min in its orbit.

In the following, some details for the set-based method are explained.[1]

The following structures of the symmetry group $\mathfrak{G}$ facilitate the computation of certainly and possibly lexicographically decreasing permutations for a specific subset. The first three structures can be preprocessed prior to the enumeration. The third structure has to be updated for each enumeration node.

Definition 2. For $n \in \mathbb{N}$ and a subgroup $\mathfrak{G}$ of $\mathfrak{S}_n$, the *hit-element classification of* $\mathfrak{G}$ is defined as

$$\text{hitclass}\colon \begin{cases} [n]\times[n] & \to & 2^{\mathfrak{G}}, \\ (i,j) & \mapsto & \{\pi \in \mathfrak{G} : \pi(i) = j\}. \end{cases} \tag{5}$$

The *increasing-element classification of* $\mathfrak{G}$ is defined as

$$\text{incclass}\colon \begin{cases} [n] & \to & 2^{\mathfrak{G}}, \\ i & \mapsto & \{\pi \in \mathfrak{G} : \pi(i) > i\} = \bigcup_{k=i+1}^{n} \text{hitclass}(i,k). \end{cases} \tag{6}$$

The *decreasing-element classification of* $\mathfrak{G}$ is defined more detailed as

$$\text{decclass}\colon \begin{cases} [n]\times[n] & \to & 2^{\mathfrak{G}}, \\ (i,j) & \mapsto & \{\pi \in \mathfrak{G} : \pi(i) < j\} = \bigcup_{k=1}^{j-1} \text{hitclass}(i,k). \end{cases} \tag{7}$$

Moreover, for a subset S of $[n]$, the *critical-element classification of* $\mathfrak{G}$ *with respect to* S is defined as

$$\text{critclass}_S\colon \begin{cases} [n]\cup\{\infty\} & \to & 2^{\mathfrak{G}}, \\ i & \mapsto & \{\pi \in \mathfrak{G} : \text{critelem}_S(\pi) = i\}. \end{cases} \tag{8}$$

The next lemma characterizes lexicographical minimality in an orbit by intersections of certain decreasing-element and critical-element classifications.

Lemma 4. *Let $n \in \mathbb{N}$ and $\mathfrak{G}$ a subgroup of $\mathfrak{S}_n$. Moreover, let S be a subset of $[n]$, and let $S^- = S \setminus \{\max S\}$ be lexicographically minimal in its $\mathfrak{G}$-orbit. Then S is not lexicographically minimal in its $\mathfrak{G}$-orbit if and only if at least one of the following cases occurs:*

I. *The set* $\text{decclass}(\max S, \max S) \cap \text{critclass}_{S^-}(\infty)$ *is non-empty.*

II. *There is an element $i \in S^-$ such that the set* $\text{decclass}(\max S, i) \cap \text{critclass}_{S^-}(i)$ *is non-empty.*

III. *There is an $i \in S^-$ and a permutation π in the set* $\text{hitclass}(\max S, i) \cap \text{critclass}_{S^-}(i)$ *such that* $\text{critelem}_S(\pi) \in \pi(S)$. □

The set-based method has the advantage that the checks that potentially lead to an immediate answer can be done prior to the complicated cases, whereas in the iteration method it depends on the order of the permutations when the complicated cases have to be handled. The disadvantage is that for large n especially the decreasing-element classification can grow large. For example, if the order of $\mathfrak{G}$ is smaller than n, then a brute-force iteration over the elements of $\mathfrak{G}$ is usually faster than handling the permutation classifications.

[1]The approach is motivated by the fact that with a set structure based on dynamic bitstrings it is possible to compute unions, intersections, differences, and symmetric differences of sets fast in practice. Nota bene: From a complexity standpoint, bitstrings only gain something if their length is uniformly bounded. However, in practice the reduction of the runtime by a constant factor by using dynamic bitstrings for set operations is not irrelevant. Moreover, operations on dynamic bitstrings residing consecutively in memory are more cache coherent than data structures that are scattered in main memory.

The applications presented in this paper differ in this respect: For enumerating triangulations (one prominent example $\Delta_6 \times \Delta_2$ has $|\mathfrak{G}| = 30{,}240$ and $n = 35{,}721$), the iteration method is mostly faster, whereas for the large cases in the enumeration of circuits and cocircuits (our largest example, the cocircuits of the 9-cube C^9, has $|\mathfrak{G}| = 185{,}794{,}560$ and $n = 512$) the set-based method is significantly faster.

The reason why the set-based method does not appear in the computational results of this paper is as follows: For large orders and small degree, an even faster method could be found based on a tighter specialization of the switch-table method. It works by combining switch tables with the recursive algorithm in [13]. The theoretical foundation of this is rather straight-forward but nevertheless useful. The recursive algorithm in [13] answers a slightly more general question: for two given subsets S_{img} and S_{org} with the same number of elements, does there exist a permutation $\pi \in \mathfrak{G}$ with $\pi(S_{\text{img}}) <_{\text{lex}} S_{\text{org}}$? The answer is certainly "no" if the subsets are empty. The answer is "yes" if the some element of S_{img} can be mapped to something smaller than the minimal element s_{min} of S_{org}. The answer is "no" if all elements of S_{img} are mapped by $\mathfrak{G}$ to something strictly larger than s_{min}. And if some $\pi \in \mathfrak{G}$ maps some element $k \in S_{\text{img}}$ exactly to s_{min}, the answer is given by answering the same question recursively for $\pi(S_{\text{img}}) \setminus \{s_{\text{min}}\}$, $S_{\text{org}} \setminus \{s_{\text{min}}\}$, and the group $\mathfrak{G}_{[s_{\text{min}}]}$. Switch tables allow to run this recursive algorithm without the computation of new stabilizer groups. Even though, computational group theory offers efficient algorithms for this, in tight loops, the allocation and deallocation of memory for new stabilizer groups can slow down the computation.[2] Our adaption is based on the following.

Lemma 5. *Consider a switch table* $\operatorname{st}(\cdot,\cdot)$ *for a subgroup* $\mathfrak{G}$ *of* $\mathfrak{S}_n$. *Let* $1 \le i \le n$, *and let* $\mathfrak{G}_{[i]}$ *be the point-wise stabilizer of* $[i]$ *in* $\mathfrak{G}$. *Moreover, let* S_{org} *and* S_{img} *be subsets of* $[n] \setminus [i-1]$. *Then:*

(i) If $S_{\text{org}} = \emptyset$, *then there is no permutation* $\pi \in \mathfrak{G}_{[i]}$ *with* $\pi(S_{\text{img}}) <_{lex} S_{\text{org}}$ *if and only if* $S_{\text{img}} <_{lex} S_{\text{org}}$.

(ii) If $\mathfrak{G}$ *is the trivial group or* $i > \max(\text{effRowSet})$, *then there is a permutation* $\pi \in \mathfrak{G}_{[i]}$ *with* $\pi(S_{\text{img}}) <_{lex} S_{\text{org}}$ *if and only if* $S_{\text{img}} <_{lex} S_{\text{org}}$.

(iii) If $i \in S_{\text{org}}$, *then:*

  (a) If $i \in S_{\text{img}}$ *and there is a permutation* $\pi' \in \mathfrak{G}_{[i+1]}$ *with* $\pi'\big(S_{\text{img}} \setminus \{i\}\big) <_{lex} S_{\text{org}} \setminus \{i\}$, *or*

  (b) if there is a non-trivial switch $\operatorname{st}(i, j_i)$ *with* $j_i \in S_{\text{img}}$ *and a permutation* $\pi' \in \mathfrak{G}_{[i+1]}$ *with* $\pi'\big(\operatorname{st}(i, j_i)(S_{\text{img}} \setminus \{j_i\})\big) <_{lex} S_{\text{org}} \setminus \{i\}$,

  then there is a permutation $\pi \in \mathfrak{G}_{[i]}$ *with* $\pi(S_{\text{img}}) <_{lex} S_{\text{org}}$. *If none of these cases occurs, then there is no permutation* $\pi \in \mathfrak{G}_{[i]}$ *with* $\pi(S_{\text{img}}) <_{lex} S_{\text{org}}$.

(iv) If $i \notin S_{\text{org}}$, *then:*

  (a) If $i \in S_{\text{img}}$, *or*

  (b) $\text{effColSet}(i) \cap S_{\text{img}} \neq \emptyset$, *or*

  (c) there is a permutation $\pi' \in \mathfrak{G}_{[i+1]}$ *with* $\pi'\big(S_{\text{img}}\big) <_{lex} S_{\text{org}}$, *or*

  (d) there is a switch $\operatorname{st}(i, j_i)$ *with* $j_i \notin S_{\text{img}}$ *and a permutation* $\pi' \in \mathfrak{G}_{[i+1]}$ *with* $\pi'\big(\operatorname{st}(i, j_i)(S_{\text{img}})\big) <_{lex} S_{\text{org}}$,

  then there is a permutation $\pi \in \mathfrak{G}_{[i]}$ *with* $\pi(S_{\text{img}}) <_{lex} S_{\text{org}}$. *If none of these three cases occurs, then there is no permutation* $\pi \in \mathfrak{G}_{[i]}$ *with* $\pi(S_{\text{img}}) <_{lex} S_{\text{org}}$.

[2] A very basic attempt to implement the original algorithm from [13] based on the publicly available `permlib` led to a substantially slower runtime than the implementation based on switch tables from [7].

In particular, for $i = 1$ and $S_{\mathrm{img}} = S_{\mathrm{org}}$ these conditions characterize whether there is a permutation $\pi \in \mathfrak{G}$ with $\pi(S_{\mathrm{org}}) <_{lex} S_{\mathrm{org}}$, i.e., whether S_{org} is not lex-min in its orbit.

Proof. There is a permutation $\pi \in \mathfrak{G}_{[i]}$ with $\pi(S_{\mathrm{img}}) <_{\mathrm{lex}} S_{\mathrm{org}}$ if and only if there is a switch product $\mathrm{st}(n, j_n)\dots\mathrm{st}(i, j_i)$ with $\big(\mathrm{st}(n, j_n)\dots\mathrm{st}(1, j_1)\big)(S_{\mathrm{img}}) <_{\mathrm{lex}} S_{\mathrm{org}}$, by [7].

Case (i) and (ii) are straight-forward.

Case (iii)(a) is formally required to account for the identity switch in row i; the argumentation is then the same as for the next case. In case (iii)(b), consider the case when there is a switch $\mathrm{st}(i, j_i)$ with $j_i \in S_{\mathrm{img}}$ and a permutation $\pi' \in \mathfrak{G}_{[i+1]}$ with $\pi'\big(\mathrm{st}(i, j_i)(S_{\mathrm{img}} \setminus \{j_i\})\big) <_{\mathrm{lex}} S_{\mathrm{org}} \setminus \{i\}$. Consider $\pi := \pi' \cdot \mathrm{st}(i, j_i)$. Then, $\pi(j_i) = i = \min(S_{\mathrm{org}})$, since $\pi' \in \mathfrak{G}_{[i+1]}$ stabilizes i. Because $j_i \in S_{\mathrm{img}}$, we have $\pi(S_{\mathrm{img}}) <_{\mathrm{lex}} S_{\mathrm{org}}$ if and only if $\pi'(S_{\mathrm{img}} \setminus \{j_i\}) <_{\mathrm{lex}} S_{\mathrm{org}} \setminus \{i\}$, which is the case by the choice of π'. If there is no switch $\mathrm{st}(i, j_i)$ with $j_i \in S_{\mathrm{img}}$, then all switch products $\mathrm{st}(n, j_n)\dots\mathrm{st}(i, j_i)$ map all elements of S_{img} to elements strictly larger than $i = \min(S_{\mathrm{org}})$, and S_{org} is therefore strictly lex-smaller than any subset in the $\mathfrak{G}_{[i]}$-orbit of S_{img}. If, moreover, for all switches $\mathrm{st}(i, j_i)$ with $j_i \in S_{\mathrm{img}}$ the $\mathfrak{G}_{[i+1]}$-orbit of $\mathrm{st}(i, j_i)(S_{\mathrm{img}} \setminus \{j_i\})$ contains no lex-smaller element than $S_{\mathrm{org}} \setminus \{i\}$, then no switch product $\mathrm{st}(n, j_n)\dots\mathrm{st}(i, j_i)$ with $j_i \in S_{\mathrm{img}}$ can lex-decrease S_{img} below S_{org}.

In case (iv)(a) the minimal element of S_{org} is strictly larger than the minimal element i of S_{img}, thus $S_{\mathrm{img}} <_{\mathrm{lex}} S_{\mathrm{org}}$ so that $\pi = \mathrm{id}$ will do the job. In case (iv)(b) there is a switch $\mathrm{st}(i, j_i)$ mapping an element of S_{img} to a smaller element than $\min(S_{\mathrm{org}}) > i$, so $\pi = \mathrm{st}(i, j_i) = \mathrm{id}\dots\mathrm{id}\cdot\mathrm{st}(i, j_i)$ maps S_{img} to a lex-smaller subset than S_{org}. The cases (iv)(c) and (d) are essentially analogous to case (iii)(a) and (b).

The final assertion follows from backward induction on i, rooted at the case $i = n$ with the trivial group $\mathfrak{G}_{[n]}$. □

Call the application of this the *modified switch-table method* for checking lexicographic minimality of a subset in its orbit. Section 5 presents an algorithm in detail using the modified switch-table method to decide for large group orders whether or not a subset is lex-min in its orbit.

5 Algorithms

In this section, the following are introduced:

- Variants of `SymLexSubsetRS` (Algorithm 5 in Section 2) supporting global and local auxiliary data, counting only maximal elements, minimal non-elements, and feasible elements, resp., and utilizing semi-deciding algorithms to prune the enumeration
- Three new ways to implement `IsLexMin(`S`, `$\mathfrak{G}$`)` in `SymLexSubsetRS(`$n, \mathscr{D}, \mathfrak{G}, S$`)`. Recall that this subroutine checks whether or not S' is lex-min in its orbit.

First, the algorithm `SymLexSubsetRS_withData(`$n, \mathscr{D}, \mathfrak{G}, \mathbf{G}, N$`)` is presented, which is a slightly modified form of `SymLexSubsetRS(`$n, \mathscr{D}, \mathfrak{G}, S$`)`. The reason is that for most non-trivial problems auxiliary data has to be computed and to be kept in memory somehow. The are two principal ways auxiliary data can be made available. Data that depends on the current subset in the reverse-search tree is contained together with the subset in a node in the reverse-search tree. Data independent of the subset can be stored globally, e.g., together with the problem data.

To keep track of this in the following, specify by $N = (S_N, \mathbf{L}_N)$ a node in the reverse-search tree consisting of a subset $S_N \in 2^{[n]}$ and a subset-specific collection of data $\mathbf{L}_N$, which formally is just a tupel of mathematical objects. The global collection of data is denoted by $\mathbf{G}$.

Any implementation of `SymLexSubsetRS_withData` must specify the exact structure of $\mathbf{L}$ and $\mathbf{G}$. It is desirable that the local data for a new node can be computed during one or both of the rather expensive subroutines IsLexMin and IsInDownset. A possible such layout can be seen in Algorithm 7.

```
Algorithm: SymLexSubsetRS_withData($n$, $\mathscr{D}$, $\mathfrak{G}$, $\mathbf{G}$, $N$)
Input: $n \in \mathbb{N}$, a downset $\mathscr{D}$ of $2^{[n]}$, a subgroup $\mathfrak{G}$ of the automorphism group of $\mathscr{D}$, global auxiliary data
        $\mathbf{G}$, a node $N = (S, \mathbf{L})$ with $S \in \mathscr{D}$ and $\mathbf{L}$ local auxiliary data
Output: the number of $\mathfrak{G}$-orbits of subsets in $\mathscr{D}$ containing $S$
/* build a depth-first-search tree with root node $N = (S, \mathbf{L})$:              */
$c \leftarrow 1$ ;                                                               /* count $S$ */
for $i = \max(S) + 1, \ldots, n$ do              /* ordered traversal of new maximal elements */
    $S' \leftarrow S \cup \{i\}$ ;                               /* add a new element on the right */
    default initialize $\mathbf{L}'$ ;                          /* prepare a local data structure */
    (answer, $\mathbf{L}'$) $\leftarrow$ IsLexMin($S', \mathfrak{G}, \mathbf{G}, \mathbf{L}, \mathbf{L}'$) ;                  /* lex-min check */
    if answer = FALSE then               /* if new set is not lex-min in its orbit */
        continue ;                                                /* next loop element */
    (answer, $\mathbf{L}'$) $\leftarrow$ IsInDownset($S', \mathscr{D}, \mathbf{G}, \mathbf{L}, \mathbf{L}'$) ;            /* membership check */
    if answer = FALSE then                                  /* if new set is not in $\mathscr{D}$ */
        continue ;                                                /* next loop element */
    $N' \leftarrow (S', \mathbf{L}')$ ;                                              /* build new node */
    $c \leftarrow c +$ SymLexSubsetRS_withData($n$, $\mathscr{D}$, $\mathfrak{G}$, $\mathbf{G}$, $N'$) ;                /* recurse */
return c;
```

Algorithm 7: A variant of symmetric lexicographic subset reverse search with explicit use of global and local auxiliary data; the global data **G** is either preprocessed prior to the first call of the algorithm with $S = \emptyset$ or lazily updated as we go along; the local data **L** is generated during the crucial subroutines; the subroutine calls have been organized on the same level so that it is easy to flip their order or even interleave their individual steps if appropriate

At times, not the cardinality of the downset $\mathscr{D}$ is of interest but only the cardinality of its maximal elements. This is, e.g., the case for two applications (cocircuits and triangulations). In such situations, the downset $\mathscr{D}$ is only implicitly defined as the downset of all subsets of the subsets one is really interested in. This models the process building interesting objects from scratch element-by-element. In the ordered reverse-search tree of subsets considered in the algorithms so far, elements are always added to subsets that are larger than the current maximal element. Therefore, the following notions are defined:

Definition 3. For a subset $S \in \mathscr{D}$ an *expansion of* S is an element $i \in [n] \setminus S$ so that $S \cup \{i\} \in \mathscr{D}$. A *right-expansion* is an expansion i with $i > \max S$. The subset S is *maximal* if there is no expansion for it, and it is *right-maximal* if there is no right-expansion for it. The subset S is *right-completable* if there is a maximal set S' that is maximal in $\mathscr{D}$ so that $i < j$ for all $i \in S$ and $j \in S' \setminus S$.

Algorithm 8 shows how to deal with situations like this. It might appear that the leaves of the enumeration tree automatically correspond to maximal subsets in $\mathscr{D}$. This, however, is unfortunately not the case – the leaves are, by construction, only right-maximal: Consider an arbitrary maximal non-empty subset S in $\mathscr{D}$. Then, e.g., the enumeration branch starting with the second smallest element in S will have $S \setminus \min S$ as one of its leaves, which is obviously not maximal in D. Thus, a maximality check has to be performed on the leaves, and non-maximal leaves are simply ignored for the count. Sometimes the reverse-search tree can be pruned by semi-deciding whether or not a subset can be right-completed. The subroutine SemiIsNotRightComp in Algorithm 8 takes care of that. Whenever it can be detected locally that an expansion is unavoidable on the path to a maximal subset, one can skip the traversal of supersets not containing it. This idea is supported by the subroutine IsInEachMax.

```
Algorithm: SymLexSubsetRSMax_withData($n, \mathscr{D}, \mathfrak{G}, \mathbf{G}, N$)
Input: $n \in \mathbb{N}$, a downset $\mathscr{D}$ of $2^{[n]}$, a subgroup $\mathfrak{G}$ of the automorphism group of $\mathscr{D}$, global auxiliary data
       $\mathbf{G}$, a node $N = (S, \mathbf{L})$ with $S \in \mathscr{D}$ and $\mathbf{L}$ local auxiliary data
Output: the number of $\mathfrak{G}$-orbits of maximal subsets in $\mathscr{D}$ containing $S$
/* build a depth-first-search tree with root node S                          */
$c \leftarrow 0$ ;                                              /* do not count S yet */
for $i = \max(S)+1, \dots, n$ do              /* ordered traversal of new maximal elements */
    $S' \leftarrow S \cup \{i\}$ ;                              /* add a new element on the right */
    default initialize $\mathbf{L}'$ ;                          /* prepare a local data structure */
    $(\text{answer}, \mathbf{L}') \leftarrow$ IsLexMin($S', \mathfrak{G}, \mathbf{G}, \mathbf{L}, \mathbf{L}'$) ;        /* lex-min check */
    if answer = FALSE then                     /* if new set is not lex-min in its orbit */
        continue ;                                              /* next loop element */
    $(\text{answer}, \mathbf{L}') \leftarrow$ IsInDownset($S', \mathscr{D}, \mathbf{G}, \mathbf{L}, \mathbf{L}'$) ;     /* membership check */
    if answer = FALSE then                                      /* if new set is not in D */
        continue ;                                              /* next loop element */
    $(\text{answer}, \mathbf{L}') \leftarrow$ SemiIsNotRightComp($S', \mathscr{D}, \mathbf{G}, \mathbf{L}, \mathbf{L}'$) ;   /* incomp. comp. check */
    if answer = TRUE then                      /* if new set is not right-completable */
        continue ;                                              /* next loop element */
    $N' \leftarrow (S', \mathbf{L}')$ ;                         /* build new node */
    $c \leftarrow c +$ SymLexSubsetRSMax_withData($n, \mathscr{D}, \mathfrak{G}, \mathbf{G}, N'$) ;     /* recurse */
    $(\text{answer}, \mathbf{L}') \leftarrow$ IsInEachMax($S', \mathscr{D}, \mathbf{G}, \mathbf{L}, \mathbf{L}'$) ;   /* check unavoidability of i */
    if answer = TRUE then                      /* if new set is not right-completable */
        break ;                                                 /* exit the loop */
if $c > 0$ then                                               /* if we found supersets in D */
    return $c$ ;                               /* return no. of max. supersets of S in D */
if IsMaxInDownset($S, \mathbf{G}, \mathbf{L}$) then                /* if S is maximal in D */
    return 1 ;                                                  /* return the count for S */
```

Algorithm 8: A variant of symmetric lexicographic subset reverse search for maximal subsets with explicit use of global and local auxiliary data; the subroutine SemiIsNotRightComp can prune the consideration of a subset if it is certainly not right-completable; the subroutine IsInEachMax checks whether each maximal superset of S contains i; the subroutine IsMaxInDownset decides whether or not the seed node is maximal in $\mathscr{D}$; since it may be an expensive check returning “FALSE” quite often, we chose to call it only for subsets that are right-maximal

Other applications are rather interested in objects that can be better represented as the minimal subsets *not* contained in a downset.

Definition 4. For a subset $R \in 2^{[n]} \setminus \mathscr{D}$ a *reduction of R* is an element $i \in R$ so that $S \setminus \{i\} \in 2^{[n]} \setminus \mathscr{D}$. The subset R is *co-minimal w.r.t.* $\mathscr{D}$ if it contains no reduction. It is *right-co-minimal* if its maximal element is not a reduction. The subset R is *right-exitable w.r.t.* $\mathscr{D}$ if there is a co-minimal set S' so that $i < j$ for all $i \in S$ and $j \in S' \setminus S$.

Our application to enumerate circuits is an example for such a use-case. The straight-forward idea for a reverse-search algorithm is to adapt SymLexSubsetRSMax_withData: add elements until the first non-member is met and check afterwards if the resulting set is a minimal non-member.

The subroutine SemiIsNotRightExit can help to prune the tree. For the enumeration of circuits as in Section 9 there is no sensible such option known, but for other application there may be one.

```
Algorithm: SymLexSubsetRSComin_withData($n, \mathscr{D}, \mathfrak{G}, \mathbf{G}, N$)
Input: $n \in \mathbb{N}$, a downset $\mathscr{D}$ of $2^{[n]}$, a subgroup $\mathfrak{G}$ of the automorphism group of $\mathscr{D}$, global auxiliary data
       $\mathbf{G}$, a node $N = (S, \mathbf{L})$ with $S \in \mathscr{D}$ and $\mathbf{L}$ local auxiliary data
Output: the number of $\mathfrak{G}$-orbits of co-minimal subsets of $\mathscr{D}$ containing $S$
/* build a depth-first-search tree with root node $S$                    */
$c \leftarrow 0$ ;                          /* as a member, $S$ cannot be co-minimal */
for $i = \max(S)+1, \dots, n$ do         /* ordered traversal of new maximal elements */
    $S' \leftarrow S \cup \{i\}$ ;                        /* add a new element on the right */
    default initialize $\mathbf{L}'$ ;                   /* prepare a local data structure */
    (answer, $\mathbf{L}'$) $\leftarrow$ IsLexMin($S', \mathfrak{G}, \mathbf{G}, \mathbf{L}, \mathbf{L}'$) ;                 /* lex-min check */
    if answer = FALSE then            /* if new set is not lex-min in its orbit */
        continue ;                                     /* next loop element */
    (answer, $\mathbf{L}'$) $\leftarrow$ IsInDownset($S', \mathscr{D}, \mathbf{G}, \mathbf{L}, \mathbf{L}'$) ;           /* membership check */
    if answer = FALSE then                 /* if new set is right-co-minimal */
        if IsCominOfDownset($S', \mathbf{G}, \mathbf{L}, \mathbf{L}'$) then          /* if set is co-minimal */
            $c \leftarrow c + 1$ ;                                        /* count $S'$ */
            continue ;                                 /* next loop element */
    (answer, $\mathbf{L}'$) $\leftarrow$ SemiIsNotRightExit($S', \mathscr{D}, \mathbf{G}, \mathbf{L}, \mathbf{L}'$) ;      /* incomp. exit. check */
    if answer = TRUE then               /* if new set is not right-exitable */
        continue ;                                     /* next loop element */
    $N' \leftarrow (S', \mathbf{L}')$ ;                                /* build new node */
    $c \leftarrow c +$ SymLexSubsetRSComin_withData($n, \mathscr{D}, \mathfrak{G}, \mathbf{G}, N'$) ;          /* recurse */
return $c$;
```

Algorithm 9: A variant of symmetric lexicographic subset reverse search for co-minimal subsets with explicit use of global and local auxiliary data; we need a subroutine IsCominOfDownset that decides whether all subsets of S are in $\mathscr{D}$, since the enumeration tree only extends subsets to the right, i.e., the first found subset not in $\mathscr{D}$ is not necessarily minimal w.r.t. arbitrary subsets; the subroutine SemiIsNotRightExit can be used to prune the reverse-search tree as soon as it is clear that $\mathscr{D}$ cannot be exited by adding elements to the right

Occasionally, the representation of interesting objects as the maximal or co-minimal elements of a downset leads to a very difficult membership test. For example, it is NP-complete to decide whether or not a set of simplices can be extended to a triangulation. See Section 10 for more details. Then, it can be preferable to travers a downset with simpler membership test that contains all feasible subsets and check for extendability to a feasible subset separately.

In the following, distinguish for a subset S of a feasible subset between an *expansion* (see Definition 3) of S by an element, which remains in $\mathscr{D}$, and an *extension* of S by an element, which remains a subset of a feasible subset. Most interesting is the case where the feasible subsets form an *antichain* in $[n]$, i.e., no feasible subset contains any other feasible subset.

Definition 5. Let $\mathscr{F}$ be an antichain in $2^{[n]}$. Members of $\mathscr{F}$ are called *feasible subsets*. Each $S \in \mathscr{F}$ *represents a solution. Extendable subsets* are subsets of feasible subsets. A subset S is *right-extendable* if it is extendable to a feasible subset S' so that $i < j$ for all $i \in S$ and $j \in S' \setminus S$.

If the membership check in the downset is simple enough, then the iteration over all new maximal elements followed by membership tests can lead to an unnecessarily large number of loop passes. In such cases it may be possible to compute all possible right-expansions of a subset before the loop and iterate only over those. The following notion supports this idea.

Definition 6. The *right-expansion sequence* $E(S)$ of S is the sequence $i_1 < \cdots < i_{|E(S)|}$ of all i_k, $k = 1,\ldots,|E(S)|$, with $S \cup \{i_k\} \in \mathscr{D}$.

For the enumeration of feasible subsets with expensive exact extendability check it is desirable to prune the enumeration tree as soon as one knows that a subset is not right-extendable. This way, the possibly expensive exact extendability checks can be restricted to promising candidates, whereas at all other enumeration nodes only incomplete checks are performed. This paper suggests an incomplete check that returns TRUE if a subset is not extendable to the right and FALSE if the subset might be extendable to the right. Algorithm 10 shows a possible pseudo-code for this method that will be used in the enumeration of all triangulations of a point configuration in Section 10.

```
Algorithm: SymLexSubsetRSFeas_withData($n, \mathscr{D}, \mathscr{F}, \mathfrak{G}, \mathbf{G}, N$)
Input: $n \in \mathbb{N}$, a downset $\mathscr{D}$ of $2^{[n]}$, a subset $\mathscr{F} \subseteq \mathscr{D}$ of feasible subsets, a subgroup $\mathfrak{G}$ of the
       automorphism group of $\mathscr{D}$, global auxiliary data $\mathbf{G}$, a node $N = (S, \mathbf{L})$ with $S \in \mathscr{D}$, and $\mathbf{L}$ local
       auxiliary data encompassing the right-expansion sequence $E(S)$
Output: the number of $\mathfrak{G}$-orbits of feasible subsets in $\mathscr{D}$ containing $S$
if IsFeasible($S, \mathscr{F}, \mathbf{G}, \mathbf{L}$) then                              /* if $S$ is feasible */
    return 1 ;                                                         /* count $S$ */
/* build a depth-first-search tree with root node $S$                          */
$c \leftarrow 0$ ;                                                      /* do not count $S$ yet */
for $j = 1,\ldots,|E(S)|$ do                         /* ordered traversal of right-expansions */
    default initialize $\mathbf{L}'$ ;                               /* prepare a local data structure */
    (break, $S'$, $\mathbf{L}'$) $\leftarrow$ Expand($S, E(S)_j, \mathbf{G}, \mathbf{L}, \mathbf{L}'$) ;          /* expand to the right inside $\mathscr{D}$ */
    if break = TRUE then                                /* if expansion returns to stop */
        break ;                                                      /* exit loop */
    (answer, $\mathbf{L}'$) $\leftarrow$ IsLexMin($S', \mathfrak{G}, \mathbf{G}, \mathbf{L}, \mathbf{L}'$) ;                        /* lex-min check */
    if answer = FALSE then                   /* if new set is not lex-min in its orbit */
        continue ;                                           /* next loop element */
    (answer, $\mathbf{L}'$) $\leftarrow$ SemiIsNotRightExt($S', \mathscr{D}, \mathscr{F}, \mathbf{G}, \mathbf{L}, \mathbf{L}'$) ;         /* incomp. ext. check */
    if answer = TRUE then              /* if new set is certainly not ext. t.t.r.  */
        continue ;                                           /* next loop element */
    $N' \leftarrow (S', \mathbf{L}')$;                                           /* build new node */
    $c \leftarrow c +$ SymLexSubsetRSFeas_withData($n, \mathscr{D}, \mathfrak{G}, \mathbf{G}, N'$) ;             /* recurse */
return $c$ ;
```

Algorithm 10: A variant of symmetric lexicographic subset reverse search for feasible subsets with pruning by an incomplete extendability check; the subroutine IsFeasible checks feasibility of a subset; the subroutine Expand computes not only the subset resulting from the union with a right-expansion, but also the expansion sequence of the new subset, stored with the node's local data; moreover, it can return "break = TRUE", if the current and the future expansions cannot lead to a feasible subset anymore; SemiIsNotRightExt semi-decides whether or not the seed node is extendable to the right, i.e., for any "TRUE" answer we know for sure that S is not extendable to the right (no need to recurse), and for any "FALSE" answer we do not know yet (we have to recurse)

Next, the findings from Section 4 are used to specify suitable local data to accelerate the lex-min check in orbits. In particular, the critical-element-table $\mathrm{critelem}_S$ of a subset will be used as local data stored together with S in a node. It can be seen that one can update the critical-element table for the next node during its usage in the lex-min check.

First, the iterative method `IsLexMin_viaIter`$(S', \mathfrak{G}, \mathbf{G}, \mathbf{L}, \mathbf{L}')$ is described, where $\mathbf{L}$ contains CET, which is a function-value table of $\mathrm{critelem}_S$. It neither needs global data $\mathbf{G}$ nor the initialized local data $\mathbf{L}'$ for S'. The updated critical-element table of S' is stored in $\mathbf{L}' := \mathsf{CET}'$ representing a function-value table of $\mathrm{critelem}_{S'}$. It iterates over the whole symmetry group but is quite lean inside the loop. Algorithm 11 shows a possible listing in pseudo-code.

```
Algorithm: IsLexMin_viaIter($S'$, $\mathfrak{G}$, CET)
Input: a set $S'$, a subgroup $\mathfrak{G}$ of $\mathfrak{S}_n$, and the critical-element table $\mathsf{CET} = \mathrm{critelem}_S$ of $S = S' \setminus \max S'$
Output: $(\mathsf{TRUE}, \mathsf{CET}' = \mathrm{critelem}_{S'})$ if $S' = \text{lex-min}\,\mathfrak{G}(S')$ and $(\mathsf{FALSE}, -)$ otherwise
$\mathsf{CET}' \leftarrow \mathsf{CET}$;                        /* copy the old critical-element table */
for $\pi \in \mathfrak{G}$ do                                          /* iterate over $\mathfrak{G}$ */
    $j \leftarrow \mathsf{CET}[\pi]$;                                /* retrieve critical element */
    if $j = \infty$ then                                          /* if $\pi$ stabilizes $S$ */
        if $\pi(\max S') < \max S'$ then                      /* if $\pi$ decreases $\max S'$ */
            return $(\mathsf{FALSE}, -)$;                           /* $\pi$ is lex-decreasing */
        else if $\pi(\max S') > \max S'$ then                 /* if $\pi$ increases $\max S'$ */
            $\mathsf{CET}'(\pi) \leftarrow \max S'$;                   /* update critical element */
    else                                               /* $\pi$ strictly lex-increases $S$ */
        if $\pi(\max S') < j$ then                      /* if $\pi$ decreases $\max S'$ beyond $j$ */
            return $(\mathsf{FALSE}, -)$;                           /* $\pi$ is lex-decreasing */
        else if $\pi(\max S') = j$ then          /* if $\pi$ maps $\max S'$ to critical element */
            $j' \leftarrow \min\big(S' \triangle \pi(S')\big)$;              /* compute critical element of $S'$ */
            if $j' \in \pi(S')$ then               /* if new critical element is in $\pi(S')$ */
                return $(\mathsf{FALSE}, -)$;                       /* $\pi$ is lex-decreasing */
            else                               /* if new critical element is not in $\pi(S')$ */
                $\mathsf{CET}'[\pi] \leftarrow j'$;                    /* update critical-element table */
return $(\mathsf{TRUE}, \mathsf{CET}')$;
```

Algorithm 11: The iteration-based critical-element method with simultaneous generation of the new critical-element table of S'; the data structures CET and CET' for the critical-element tables of S and S', resp., must be stored with the subset S and S' during the enumeration

Next, the set-based variant `IsLexMin_viaSets`$(S', \mathfrak{G}, \mathbf{G}, \mathbf{L}, \mathbf{L}')$ is described, where $\mathbf{L}$ contains CEC, which is a function-value table of $\mathrm{critclass}_S$. Global data $\mathbf{G}$ containing $(\mathsf{HEC}, \mathsf{IEC}, \mathsf{DEC})$ are needed, which are function-value tables for hitclass, incclass, and decclass, respectively. It does not need the initialized local data $\mathbf{L}'$ for S'. Algorithm 12 shows the pseudo-code for this.

Whenever the group order becomes very large, then a generator-based method can become necessary. Using Lemma 5 from Section 4 Algorithm 13 proposes a new method that combines the ideas from [7] with the ideas from [13]. The advantage compared to Algorithm 6 is that it recursively removes elements from the subsets that play no role in the lex-comparison. The advantage compared to the algorithm in [13] is that one only uses a single group representation, namely a switch table.

```
Algorithm: IsLexMin_viaSets($S', \mathfrak{G}, (\mathsf{HEC}, \mathsf{IEC}, \mathsf{DEC}), \mathsf{CEC}$)
Input: a set $S'$, a subgroup $\mathfrak{G}$ of $\mathfrak{S}_n$ represented by its hit-element, increasing-element, and
       decreasing-element classifications HEC, IEC, and DEC, the critical-element classification
       $\mathsf{CEC} = \mathrm{critclass}_S$ of $S = S' \setminus \max S'$
Output: $(\mathsf{TRUE}, \mathsf{CEC}' = \mathrm{critclass}_{S'})$ if $S' = $ lex-min $\mathfrak{G}(S')$ and $(\mathsf{FALSE}, -)$ otherwise
if $\mathsf{DEC}[\max S'][\max S'] \cap \mathsf{CEC}[\infty] \neq \emptyset$ then                /* check for case I */
    return $(\mathsf{FALSE}, -)$ ;                                          /* $S'$ is not lex-min */
if $\exists i \in S : \mathsf{DEC}[\max S'][i] \cap \mathsf{CEC}[i] \neq \emptyset$ then                /* check for case II */
    return $(\mathsf{FALSE}, -)$ ;                                          /* $S'$ is not lex-min */
$\mathsf{CEC}' \leftarrow \mathsf{CEC}$ ;                  /* copy the old critical-element classification */
for $j \in S$ do                                                          /* iterate over $S$ */
    for $\pi \in \mathsf{CEC}[j] \cap \mathsf{HEC}[\max S'][j]$ do                            /* check $j$ for case III */
        $j' \leftarrow \min\bigl(S' \triangle \pi(S')\bigr)$ ;                       /* compute critical element of $S'$ */
        if $j' \in \pi(S')$ then                              /* if new critical element is in $\pi(S')$ */
            return $(\mathsf{FALSE}, -)$ ;                                  /* $\pi$ is lex-decreasing */
        else                                        /* if new critical element is not in $\pi(S')$ */
            $\mathsf{CEC}'[j] \leftarrow \mathsf{CEC}'[j] \setminus \{\pi\}$ ;                           /* update classification */
            $\mathsf{CEC}'[j'] \leftarrow \mathsf{CEC}'[j'] \cup \{\pi\}$ ;                          /* update classification */
for $\pi \in \mathsf{IEC}[\max S'] \cap \mathsf{CEC}[\infty]$ do             /* for permutations no longer stabilizing */
    $\mathsf{CEC}'[\infty] \leftarrow \mathsf{CEC}'[\infty] \setminus \{\pi\}$ ;                          /* update classification */
    $\mathsf{CEC}'[\max S'] \leftarrow \mathsf{CEC}'[\max S'] \cup \{\pi\}$ ;                     /* update classification */
return $(\mathsf{TRUE}, \mathsf{CEC}')$;
```

Algorithm 12: The set-based critical-element method with simultaneous generation of the new critical-element classification of S'; the data structures CEC and CEC$'$ for the critical-element classification of S and S', resp., must be stored as local data with the subset S and S' during the enumeration; the data structures HEC, IEC, and DEC do not depend on the subset and are therefore global data; the advantage is that the easy cases I and II are checked prior to the difficult case III from Lemma 4; fast implementations of set-operations are mandatory

```
Algorithm: IsLexMin_viaSwitchesMod($i, S, S', \mathfrak{G}, \mathsf{ST}$)
Input: an integer $i \in [1,n]$, a subset $S$ (the original subset), a subset $S'$ (the subset mapped by a partial
        switch product), a subgroup $\mathfrak{G}$ of $\mathfrak{S}_n$, and a switch table $\mathsf{ST}$ of $\mathfrak{G}$
Output: TRUE if $S = \text{lex-min}\,\mathfrak{G}(S)$ and FALSE otherwise
/* check for empty set:                                                    */
if $S = \emptyset$ then
    return TRUE;
if $\mathfrak{G}$ trivial then
    return $(S' \not<_{\text{lex}} S)$;
/* are we beyond the effective row set?                                    */
if $i > \max \text{effRowSet}(\mathsf{ST})$ then
    return $(S' \not<_{\text{lex}} S)$;
/* case distinction whether $i \in S$:                                       */
if $i \in S$ then
    if $i \in S'$ then
        /* recursively check a switch product with leading identity:       */
        if IsLexMin_viaSwitchesMod($i+1, S \setminus \{i\}, S' \setminus \{i\}, \mathfrak{G}, \mathsf{ST}$) = FALSE then
            return FALSE;
    /* travers all switches mapping an element of $S'$ to $i$:                  */
    for $j \in \text{effColSet}(i) \cap S'$ do
        if IsLexMin_viaSwitchesMod($i+1, S \setminus \{i\}, \mathsf{ST}[i][j](S' \setminus \{j\}), \mathfrak{G}, \mathsf{ST}$) = FALSE then
            return FALSE;
    return TRUE ;                    /* no decreasing switch product found */
else
    /* compare minimal elements using $\min(S) > i$:                             */
    if $i \in S'$ then
        return FALSE;
    /* check for a switch mapping an element of $S'$ to $i$:                    */
    if $\text{effColSet}(i) \cap S' \neq \emptyset$ then
        return FALSE;
    /* recursively check a switch product with leading identity:           */
    if IsLexMin_viaSwitchesMod($i+1, S, S', \mathfrak{G}, \mathsf{ST}$) = FALSE then
        return FALSE;
    /* travers all switches mapping a non-element of $S'$ to $i$:               */
    for $j \in \text{effColSet}(i) \setminus S'$ do
        if IsLexMin_viaSwitchesMod($i+1, S, \mathsf{ST}[i][j](S'), \mathfrak{G}, \mathsf{ST}$) = FALSE then
            return FALSE;
    return TRUE ;                    /* no decreasing switch product found */
```

Algorithm 13: The modified switch-table method using a combination of the specialized switch-table method of Algorithm 6 based on [7] with the recursive method in [13]; it checks whether a subset S' cannot be mapped to a lex-smaller subset than S by the permutations in $\mathfrak{G}$ stabilizing the elements $\{1,\ldots,i-1\}$; for $i=0$ and $S'=S$, it checks whether S is lex-min in its $\mathfrak{G}$-orbit; its global data is a switch table for $\mathfrak{G}$

6 Applications: Common Preliminaries

In this section, some basic notions and notation are summarized for the common setup of the applications presented in the upcoming sections.

Consider a point or vector configuration of rank r with n points or vectors. It is represented by an $r \times n$-matrix $\mathbf{A}$ containing the (homogeneous) coordinates $\mathbf{a}_1, \dots, \mathbf{a}_n \in \mathbb{R}^r$ of the points or vectors as columns. For a subset S of column indices of $\mathbf{A}$ denote by $\mathbf{A}_{*,S}$ the submatrix consisting of the columns with indices in S. Moreover, for a matrix $\mathbf{M}$ denote by $\mathbf{M}_{\mathrm{NZ}}$ its submatrix of non-zero columns.

The usual language of linear algebra can be adapted to subsets.

Definition 7. A subset $B \in [n]$ is *spanning* if $\operatorname{rank}(\mathbf{A}_{*,B}) = r$, otherwise it is *not-spanning* or *coplanar.* A coplanar subset B with rank $B = r-1$ is *hyperplanar.* It is *independent* if $\operatorname{rank}(\mathbf{A}_{*,B}) = |B|$, otherwise it is *dependent.* A *basis* (or a *simplex* in the context of triangulations) is an independent spanning subset. An $r-1$-subset of a simplex is a *simplex facet* or simply *facet* whenever no confusion with facets of $\mathbf{A}$ can arise.

The first application deals with cocircuits, which can be seen as hyperplanes spanned by the configuration.

Definition 8. Any inclusion-maximal coplanar subset C_0^* of column indices of the configuration $\mathbf{A}$ is called *(the zero part of) a cocircuit.* A *cocircuit signature* of C_0^* is a map $\sigma^*\colon [n] \to \{-,0,+\}$ with $C_0^* = \sigma^{-1}(\{0\})$ so that there is a $\mathbf{c} \in \mathbb{R}^r$ with $\mathbf{c}^T\mathbf{a}_i = 0$ for all $i \in (\sigma^*)^{-1}(0)$, $\mathbf{c}^T\mathbf{a}_i > 0$ for all $i \in (\sigma^*)^{-1}(+)$, and $\mathbf{c}^T\mathbf{a}_i < 0$ for all $i \in (\sigma^*)^{-1}(-)$. Here, $C_+^* := (\sigma^*)^{-1}(+)$ is called the *positive part,* $C_-^* := (\sigma^*)^{-1}(-)$ the *negative part* of the cocircuit signature σ^*. By elementary linear algebra, there are exactly two *opposite* cocircuit signatures of C_0^*. Moreover, these two signatures are uniquely determined by any hyperplanar subset of C_0^*. The pair (C_+^*, C_-^*) is a *signed cocircuit.*

Intuitively, a cocircuit is the subset of all elements lying on a hyperplane spanned by some of the elements in $\mathbf{A}$. Counting all cocircuits is, therefore, the same as counting all hyperplanes spanned by elements of the configuration.

Another application is concerned with circuits, which can be seen as intersection points of subspaces spanned by the configuration.

Definition 9. Any inclusion-minimal dependent subset C of the column indices of $\mathbf{A}$ is *(the support of) a circuit.* A *circuit signature* of a circuit C is a map $\sigma\colon [n] \to \{-,0,+\}$ so that $C = \sigma^{-1}(\{-,+\})$ and $\sum_{i\in\sigma^{-1}(+)} \lambda_i \mathbf{a}_i = \sum_{i\in\sigma^{-1}(-)} \lambda_i \mathbf{a}_i$ for suitable $\lambda_i > 0$, $i = 1, 2, \dots, n$. Here, $C_+ := \sigma^{-1}(+)$ is called the *positive part,* $C_- := \sigma^{-1}(-)$ the *negative part,* and $C_0 := \sigma^{-1}(0)$ the *zero-part* of the circuit signature σ. By elementary linear algebra, there are exactly two *opposite* circuit signatures of C. The pair (C_+, C_-) is a *signed circuit.*

The third application in this paper deals with triangulations of $\mathbf{A}$. It is advantageous to use a well-known characterization of triangulations of point configurations as the definition [5, Cor. 4.1.32]. In the context of triangulations, bases are usually called *simplices.*

Definition 10. Two simplices S_1 and S_2 are *intersecting properly* if there is no signed circuit C with $C_+ \subseteq S_1$ and $C_- \subseteq S_2$. A simplex facet F is *interior* if there is a cocircuit C_0^* with $F \subseteq C_0^*$ so that C_+^* and C_-^* are non-empty. A non-empty subset $\mathcal{T}$ of simplices is *covering* if for each interior facet F of some simplex $S \in \mathcal{T}$ there is another simplex $S' \in \mathcal{T}$ containing F. A *triangulation* is a non-empty covering subset $\mathcal{T}$ of pairwise properly intersecting simplices.

Proper intersection of two simplices S_1 and S_2 roughly means geometrically that the convex hulls $\operatorname{conv}\mathbf{A}_{*,S_1}$ and $\operatorname{conv}\mathbf{A}_{*,S_2}$ intersect in a common (possibly empty) face. The covering property roughly means geometrically that no interior facet of a simplex is uncovered.

7 Applications: Computational Environment

For all computational tests a C++-implementation in `TOPCOM` version `1.0.18` was used, which is essentially equivalent to `TOPCOM` version `1.1.0` that is distributed together with this paper. See [14] for a paper on an earlier version of `TOPCOM`.

Two computers were used to generate the computational results. The main computer was a Intel(R) Xeon(R) CPU E5-2690, 2.90GHz (384 GB RAM) with 2 sockets of 8 cores each and two virtual threads per core. The operating system was `Ubuntu Linux 4.15.0-162-generic` with the C++compiler `gcc (Ubuntu 7.5.0-3ubuntu1 18.04) 7.5.0`. Unless state otherwise, for each run 16 threads were used since hyper-threading has no advantages for computationally demanding tasks.

Triggered by a downtime of this machine over several months, also an Apple MacBookPro (2021) was used with M1Max (64 GB RAM) with 8 performance cores and 2 efficiency cores. The operating system was `MacOSX Monterey 12.3.1` with the C++-compiler `Apple clang version 13.1.6.` Unless stated otherwise, this computer was run with 8 threads. Computational results achieved with this computer are marked with “M1Max”.

8 Application I: Cocircuits

In this section, specializations of the subroutines in Algorithm 8 are specified for the enumeration of cocircuits of a (point or vector) configuration. The idea is as follows:

- As the downset $\mathscr{D}$ take the downset of all coplanar subsets S.
- Its maximal subsets correspond to the (zero sets of) cocircuits.
- For subsets S in $\mathscr{D}$ that are right-maximal, first check whether the corresponding columns of $\mathbf{A}$ span a subspace of rank $r-1$. If no, the subset is not maximal in $\mathscr{D}$. If yes, compute its (up to sign-reversal) unique signature. The subset is then maximal in $\mathscr{D}$ if and only if no element outside S has a zero-signature.
- In order to prune the search, check for each subset
  - whether an earlier element is contained in the generated space;
  - whether the remaining elements are sufficient to generate a space of corank one.

In order to make this idea for an application of `SymLexSubsetRSMax_withData` explicit, its subroutines `IsInDownset`, `SemiIsNotRightComp`, `IsInEachMax`, and `IsMaxInDownset` must be formally specified. Recall that in these subroutines one can utilize global data (preprocessed prior to the enumeration) and local data (updated in each enumeration node) to speed up the computations and avoid duplicate work.

In this particular case, the part of the local data $\mathbf{L}$ used by `IsMaxInDownset` encompasses a matrix M that is a column-echelon form of $\mathbf{A}_S$. It can be computed by Gaussian elimination on columns from left to right. Storing it allows us to avoid the repetition of identical eliminations in matrices with identical initial segments of columns. The local data of the node of a subset S' with $S = S' \setminus \max S$ is initialized as the augmented matrix $(\mathsf{M}, \mathbf{A}_{*,\max S})$. The global data used is the full matrix $\mathbf{A}$.

The implementation of `SemiIsNotRightComp` in Algorithm 15 is based on the following theorem.

Theorem 1. *Let S be right-completable to a (the zero-set of a) cocircuit C_0^* of* $\mathbf{A}$. *Then:*

```
Algorithm: IsInDowns‍et($S', \mathscr{D}, \mathbf{A}, \mathsf{M}, \mathsf{M}'$)
Input: A subset $S'$ of column indices of $\mathbf{A}$, the downset $\mathscr{D}$ of coplanar subsets, the configuration $\mathbf{A}$, a
    column-echelon form $\mathsf{M}$ of $\mathbf{A}_{*,S'\setminus\max S'}$, and the augmented matrix $\mathsf{M}' = (\mathsf{M}, \mathbf{A}_{*,\max S'})$ (not yet in
    column-echelon form, in general)
Output: ($\mathsf{TRUE}, \mathsf{M}'$) with $\mathsf{M}'$ a column-echelon form of $\mathbf{A}_{*,S'}$ if $\operatorname{rank}(\mathsf{M}') \le r-1$ and ($\mathsf{FALSE}, \mathsf{M}'$)
    otherwise
$\mathsf{M}' \leftarrow$ colechForm($\mathsf{M}'$);                       /* compute column-echelon form */
if $|S'| < r$ or $\mathsf{M}'_{*,r} = \mathbf{0}$ then                 /* if at most $r-1$ non-zero columns */
    return ($\mathsf{TRUE}, \mathsf{M}'$);                                        /* $S$ is coplanar */
else
    return ($\mathsf{FALSE}, \mathsf{M}'$);                                       /* $S$ is spanning */
```

Algorithm 14: Check whether a subset of columns is coplanar; the local data M' of S' is computed here; because the local data M of S is already in column-echelon form, the subroutine `colechForm` needs to update at most $\operatorname{rank}(\mathsf{M})$ many values in the right-most column of $(\mathsf{M}, \mathbf{A}_{\max S'})$, possibly after a single swap of two columns

1. *For the set* $R := \{i \in [n] : i > \max S\}$ *the rank bound* $\operatorname{rank}(\mathbf{A}_{*,S\cup R}) \ge r-1$ *holds.*
2. *For all* $i \in [n] \setminus S$ *with* $i < \max S$ *the rank bound* $\operatorname{rank}(\mathbf{A}_{*,S}) < \operatorname{rank}(\mathbf{A}_{*,S\cup\{i\}})$ *holds.*

Proof. For item 1 assume that S is right-completable to a maximal and, thus, maximally hyperplanar subset S' with $\operatorname{rank}(\mathbf{A}_{*,S\cup R}) < r-1$. Then, since $S' \setminus S \subseteq R$, we have $\operatorname{rank}(\mathbf{A}_{*,S'}) < r-1$, contradicting the fact that S' is hyperplanar.

For item 2 assume there is an $i \in [n] \setminus S$ with $i < \max S$ so that $\operatorname{rank}(\mathbf{A}_{*,S}) = \operatorname{rank}(\mathbf{A}_{*,S\cup\{i\}})$, and assume there is a maximal hyperplanar subset S' that is a right-completion of S. Then, i is not in S', and $\operatorname{rank}(\mathbf{A}_{*,S'\cup\{i\}}) = \operatorname{rank}\big(\mathbf{A}_{*,(S'\setminus S)\cup(S\cup\{i\})}\big) = \operatorname{rank}\big(\mathbf{A}_{*,(S'\setminus S)\cup S}\big) = \operatorname{rank}(\mathbf{A}_{*,S'})$, contradicting the maximality of S'. □

Call the use of the semi-check based on this theorem *rank-pruning*; skipping this semi-check is called *no-pruning* for later reference. In Table 1 the node counts for no-pruning and rank-pruning are compared for tiny to small examples.

In order to find out whether an expansion is in each maximal superset, one can use a similar observation: any element that does not increase the rank of the current subset can be added to each rank-$(r-1)$ subset without increasing the rank. Thus, it is in each maximal superset. The pseudocode is listed in Algorithm 16. All data necessary for this check has been computed before. Thus, this check is very fast, and there is no point in skipping it.

In order to implement the maximality check one has to compute a signature corresponding to a hyperplanar subset. This can be done as follows.

Lemma 6. *Let S be a hyperplanar subset and let $\mathbf{B}$ be the $r \times (r-1)$-matrix of non-zero-columns of a column-echelon form of $\mathbf{A}_{*,S}$. Then, there is a unique cocircuit C_0^* containing S. Moreover, one of the two opposite signatures of C_0^* is given by*

$$\sigma_S^*(i) = \operatorname{sign}\big(\det(\mathbf{B}, \mathbf{a}_i)\big). \tag{9}$$

In particular, S is maximal and, thus, a cocircuit if and only if $\det(\mathbf{B}, \mathbf{a}_i) \neq 0$ for all $i \in [n] \setminus S$.

Proof. The assertion follows from the fact that the right-hand side is the sign of a linear form in $\mathbf{a}_1, \dots, \mathbf{a}_r$ that vanishes on all points in $\mathbf{A}_{*,S}$, which was assumed to be hyperplanar. □

```
Algorithm: SemiIsNotRightComp($S', \mathscr{D}, \mathbf{A}, \mathsf{M}, \mathsf{M}'$)
Input: A subset $S'$ of column indices of $\mathbf{A}$, the downset $\mathscr{D}$ of coplanar subsets, the configuration $\mathbf{A}$, a
       column-echelon form $\mathsf{M}$ of $\mathbf{A}_{*,S'\setminus\max S}$, and a column-echelon form $\mathsf{M}'$ of the augmented matrix
       $\mathbf{A}_{*,S'}$
Output: ($\mathsf{TRUE}, \mathsf{M}'$) if $S'$ cannot be right-completed and ($\mathsf{FALSE}, \mathsf{M}'$) otherwise
$r' \leftarrow \operatorname{rank}(\mathsf{M}')$ ;                                  /* get current rank */
if $r' > r-1$ then                                           /* if rank too large */
    return ($\mathsf{TRUE}, \mathsf{M}'$) ;                           /* $S'$ not right-completable */
else                                                     /* if rank not too large */
    for $i = 1, \ldots, \max S' - 1$ with $i \notin S'$ do           /* traverse left non-elements */
        $\mathsf{M}'' \leftarrow$ colechForm($\mathsf{M}', \mathbf{a}_i$) ;                /* compute column-echelon form */
        if $\mathsf{M}''_{*,r'+1} = \mathbf{0}$ then                    /* if column $i$ is in the current span */
            return ($\mathsf{TRUE}, \mathsf{M}'$) ;                       /* $S'$ not right-completable */
    if $r' < r-1$ then                                      /* if rank not yet sufficient */
        $\mathsf{M}'' \leftarrow \mathsf{M}'$ ;                          /* prepare a rank-increase checker matrix */
        $r'' \leftarrow r'$ ;                                       /* keep track of rank increase */
        for $i = \max S' + 1, \ldots, n$ do                    /* traverse right non-elements */
            $\mathsf{M}'' \leftarrow$ colechForm($\mathsf{M}'', \mathbf{a}_i$) ;           /* compute column-echelon form */
            if $\mathsf{M}''_{*,r''+1} \neq \mathbf{0}$ then                            /* if $i$ increases rank */
                $r'' \leftarrow r'' + 1$ ;
                if $r'' = r-1$ then                          /* if rank increase sufficient */
                    return ($\mathsf{FALSE}, \mathsf{M}'$) ;            /* $S'$ might be right-completable */
            else if $r'' + n - i < r - 1$ then               /* if target rank unreachable */
                return ($\mathsf{TRUE}, \mathsf{M}'$) ;                   /* $S'$ not right-completable */
    return ($\mathsf{FALSE}, \mathsf{M}'$) ;                          /* $S'$ might be right-completable */
```

Algorithm 15: Semi-check whether a subset of columns is right-completable; we use rank computations that detect whenever adding columns to the right cannot reach the required rank and to detect whenever a column to the left does not increase the span; because M' is already in column-echelon form, each column-echelon form requires the computation of at most r'' new entries plus possibly a single swap of two columns; the rank is then simply the number of non-zero columns; the local data M' remains unchanged

$\mathbf{A}$	# cocircuits	# nodes by pruning method		CPU time [s] by pruning method	
	in total	no	rank	no	rank
C^5	3254	1,026,635	78,049	13.62	3.37
$\Delta_4 \times \Delta_4$	460	2,018,569	87,439	37.52	5.36
$\Delta(8,2)$	1661	3,133,113	131,006	50.88	6.60

Table 1: Comparison of no-pruning versus rank-pruning on tiny to small examples; symmetry handling has been deactivated, and only a single thread has been used in order to focus on the effectivity and efficiency of pruning alone; rank-pruning significantly reduces both the number of enumeration nodes and the computation times

```
Algorithm: IsInEachMax($S'$, $\mathscr{D}$, $\mathbf{A}$, M, M')
Input: A subset $S'$ of column indices of $\mathbf{A}$, the downset $\mathscr{D}$ of coplanar subsets, the configuration $\mathbf{A}$, a
        column-echelon form M of $\mathbf{A}_{*,S'\setminus\max S'}$, and a column-echelon form M' of the augmented matrix
        $\mathbf{A}_{*,S'}$
Output: (TRUE, M') if each maximal superset of $S$ contains $S'$ and (FALSE, M') otherwise
if rank(M') = rank(M) then                               /* $S'$ did not increase rank */
    return (TRUE, M');
else
    return (FALSE, M');
```

Algorithm 16: Check whether the expansion from S to S' is in each maximal superset of S; the local data M′ remains unchanged

That is, whenever during the enumeration a right-maximal hyperplanar subset is reached, the first $r-1$ non-zero columns M_{NZ} of the corresponding matrix in the local data determine the signature of the unique cocircuit containing it.

```
Algorithm: IsMaxInDownset($S$, $\mathscr{D}$, $\mathbf{A}$, M)
Input: A subset $S$ of column indices of $\mathbf{A}$, the downset $\mathscr{D}$ of coplanar subsets, the configuration $\mathbf{A}$, and
        a column-echelon form M of $\mathbf{A}_{*,S}$
Output: TRUE if $S$ is maximal in $\mathscr{D}$ and FALSE otherwise
for $i \in [n] \setminus S$ do                           /* for all elements outside $S$ */
    if det($\mathsf{M}_{\mathrm{NZ}}$, $\mathbf{a}_i$) = 0 then     /* if the signature of $i$ is zero */
        return FALSE;                                    /* $S$ is not maximal */
return TRUE;
```

Algorithm 17: Check whether a subset of columns is maximally coplanar; the determinant calculation requires the elimination of at most r values in the right-most column

Next, some computational results are presented. Particularly interesting is the enumeration of hyperplanes spanned by the vertices of the d-dimensional hypercube C^d. This problem has already been studied a long time ago by Aichholzer and Aurenhammer [1]. Using cleverly a lot of structural properties of hypercubes in particular, they were able to enumerate all hyperplanes spanned by the vertices of the 8-cube, while the 9-cube's hyperplanes remained out-of-reach. In contrast to their efforts, the general-purpose algorithm of this paper could compute their numbers without using any specific knowledge about cubes. And it was able to compute the number of cocircuits (total and up to symmetry) of the 9-cube. Table 2 shows the results.

9 Application II: Circuits

The idea to utilize `SymLexSubsetRS_withData` from Algorithm 7) for the enumeration of all circuits of a point configuration **A** is to utilize `SymLexSubsetRSComin_withData` on the downset of all linearly independent column subsets of **A**.

In order to use `SymLexSubsetRSComin_withData`, one has to specify how its problem-specific subroutines `IsInDownset`, `SemiIsNotRightExit`, and `IsCominOfDownset` work. Again, in these subroutines we global data (preprocessed prior to the enumeration) and local data (updated in each enumeration node) can be utilized to speed up the computations and avoid duplicate work. This

$\mathbf{A}$	# symmetries	# cocircuits up to symmetry	# cocircuits in total	# nodes	CPU time [hh:mm:ss]
C^2	8	2	6	8	0:00:00
C^3	48	3	20	30	0:00:00
C^4	384	6	140	136	0:00:00
C^5	3840	15	3254	760	0:00:00
C^6	46,080	63	252,434	7125	0:00:01
C^7	645,120	623	71,343,208	249,455	0:00:04
C^8	10,321,920	22,432	86,246,755,608	2,403,057	0:03:02
*C^9	185,794,560	3,899,720	448,691,419,804,586	530,623,380	13:30:12
*$\Delta(8,3)$	40,320	56	166,420	4,643	0:00:00
*$\Delta(8,4)$	40,320	144	1,105,575	14,030	0:00:00
*$\Delta(9,3)$	362,880	231	10,004,154	21,033	0:00:02
*$\Delta(9,4)$	362,880	2,522	359,022,180	226,076	0:00:07
*$\Delta(10,3)$	3,628,800	1,337	889,205,792	113,813	0:00:21
*$\Delta(10,4)$	3,628,800	87,254	178,227,172,388	6,889,143	0:07:38
*$\Delta(10,5)$	3,628,800	387,983	939,079,703,204	41,451,047	1:47:49

Table 2: Computational results for the enumeration of cocircuits in hypercubes and hypersimplices using 16 threads; the total numbers for hypercubes up to dimension 8 independently confirm the results in [1], while the numbers up to symmetry and dimension 9 are new as well as all the numbers for hypersimplices (marked with a "*"), to the best of my knowledge; for C^9 compare the total number of its cocircuits to the number of all its $r-1$-subsets, which is $\binom{512}{9} = 6{,}208{,}116{,}950{,}265{,}950{,}720$ (four orders of magnitude larger) – direct signature computations for all these subsets, given today's computation power, would have been out of reach by far

time the local data will be a matrix in column-echelon form stacked on top of another matrix. The columns of the additional matrix yield additional information useful for circuits. For a subset S the columns of the top matrix are the columns of a column-echelon form of the sub configuration $\mathbf{A}_{*,S}$. The column in the bottom matrix contains the coefficients of a linear combination of all original columns weakly to the left that yields the column on top. If the column on top is the zero-column, then the corresponding original column can be combined linearly from original columns strictly to the left, and the signs of the coefficients in the bottom column yield the corresponding circuit signature.

Formally, the stacked matrix can be defined as follows:

Definition 11. Let $\mathbf{A}_{*,S}$ be a subset of columns of a rank-r configuration $\mathbf{A}$. For an integer $k \in [1, r+1]$ let $\mathbf{I}_k$ denote the $k \times k$-identity matrix.

A $(r + |S|) \times |S|$-matrix $\mathbf{R} = \binom{\mathbf{B}}{\mathbf{C}}$ is a *column-representation matrix of S* if it is a column-echelon form of the matrix

$$\begin{pmatrix} \mathbf{A}_{*,S} \\ \mathbf{I}_{|S|} \end{pmatrix} \tag{10}$$

Here, $\mathbf{B}$ is the *configuration part*, whereas $\mathbf{C}$ is the *coefficient part*.

Theorem 2. *Let $\mathbf{R} = \binom{\mathbf{B}}{\mathbf{C}}$ be a column-representation matrix of a subset S of column indices of $\mathbf{A}$. Then S is dependent if and only if $\mathbf{B}_{*,|S|} = \mathbf{0}$. Moreover, S is a circuit if and only if $\mathbf{B}_{*,|S|}$ is the first zero-column in $\mathbf{B}$, and $\mathbf{C}_{|S|}$ contains no zero entry. In that case, the signs in $\mathbf{C}_{|S|}$ specify one of the two possible circuits signatures of S restricted to its support S.*

Proof. Observe that

$$\begin{pmatrix} \mathbf{A}_{*,S} \\ \mathbf{I}_{|S|} \end{pmatrix} \cdot \mathbf{I}_{|S|} = \begin{pmatrix} \mathbf{A}_{*,S} \\ \mathbf{I}_{|S|} \end{pmatrix}. \tag{11}$$

If this is transformed by admissible column operations, represented by the multiplication of a matrix **C** from the right, into column-echelon form, one has:

$$\begin{pmatrix} \mathbf{A}_{*,S} \\ \mathbf{I}_{|S|} \end{pmatrix} \cdot \mathbf{C} = \begin{pmatrix} \mathbf{B} \\ \mathbf{C} \end{pmatrix}. \tag{12}$$

with column-representation matrix $\binom{\mathsf{B}}{\mathsf{C}}$ of S. In particular, one has for the last column:

$$\mathbf{A}_{*,S} \cdot \mathbf{C}_{*,|S|} = \mathbf{B}_{*,|S|}. \tag{13}$$

S is dependent if and only if the last column is zero, by the properties of a column-echelon form. Moreover, the entries of $\mathbf{C}_{*,|S|}$ constitute a linear dependence. Since the first rank $\mathbf{B}$ columns of $\mathbf{B}$ are linearly independent and $\mathbf{B}_{*,|S|}$ is the first zero-column in $\mathbf{B}$, one has that $\operatorname{rank}(\mathbf{A}_{*,S}) = |S| - 1$. Thus, the kernel of $\mathbf{A}_{*,S}$ is one-dimensional, and a non-zero vector in it is unique up to a non-zero scalar multiple. Therefore, the signs of $\mathbf{C}_{*,|S|}$ are unique up to sign-reversal. Thus, the non-zero components of $\mathbf{C}_{*,|S|}$ constitute the inclusion minimal support of a linear dependence among the columns in $\mathbf{A}_{*,S}$. Hence, the signs of the components of the complete vector $\mathbf{C}_{*,|S|}$ are a circuit signature on S if and only if there are no zero-components in it. □

One can derive all necessary information to process a node from the column-representation matrix of its subset. For the membership in the downset of independent sets this is straight-forward. Algorithm 18 shows the procedure. A useful semi-check for right-exitability of a subset is still an open

```
Algorithm: IsInDownset($S'$, $\mathscr{D}$, $\mathbf{A}$, R, R$'$)
Input: A subset $S'$ of column indices of $\mathbf{A}$, the downset $\mathscr{D}$ of coplanar subsets, the configuration $\mathbf{A}$, a
       column-representation matrix $\mathsf{R} = \binom{\mathsf{B}}{\mathsf{C}}$ of $S'$, and the augmented matrix $\mathsf{R}' = \left(\mathsf{R}, \binom{\mathbf{A}_{*,\max S'}}{\mathbf{e}_{|S'|}}\right)$ (not
       yet in column-echelon form, in general), where $\mathbf{e}_{|S'|}$ is the unit vector in $\mathbb{R}^{|S'|}$ with a one in its
       last component
Output: (TRUE, R$'$) with $\mathsf{R}' = \binom{\mathsf{B}'}{\mathsf{C}'}$ a column-representation matrix of $\mathbf{A}_{*,S'}$ if rank(R$'$) = $|S'|$ and
        (FALSE, R$'$) otherwise
R$'$ ← colechForm(R$'$) ;                          /* compute column-echelon form */
if $\mathsf{B}'_{*,|S'|} \neq \mathbf{0}$ then                     /* if right-most conf-column non-zero */
    return (TRUE, R$'$) ;                                   /* $S'$ is independent */
else
    return (FALSE, R$'$) ;                                    /* $S'$ is dependent */
```

Algorithm 18: Check whether a subset of columns is independent; the local data R$'$ of S' is computed here; because the local data R of S is already in column-echelon form, the subroutine colechForm needs to update at most 2 rank(R) many values in the right-most column of $\left(\mathsf{R}, \binom{\mathbf{A}_{*,\max S'}}{\mathbf{e}_{|S'|}}\right)$, possibly after a single swap of two columns

problem. Thus, for circuit enumeration SemiIsNotRightExit simply returns FALSE for each subset (i.e., the subset might be a left segment of a co-minimal subset of $\mathscr{D}$).

Given Lemma 2 it is easy to check a subset for co-minimality. The corresponding algorithm is listed in Algorithm 19.

Table 3 shows some results that could be obtained using the resulting algorithm.

```
Algorithm: IsCominOfDownset($(S', \mathscr{D}, \mathbf{A}, \mathsf{R}, \mathsf{R}')$)
Input: A subset $S$ of column indices of $\mathbf{A}$, the downset $\mathscr{D}$ of independent subsets, the configuration $\mathbf{A}$,
       and local data given by a column-representation matrix $\mathsf{R}' = \binom{\mathsf{B}'}{\mathsf{C}'}$ of $\mathbf{A}'_{*,S}$
Output: TRUE if $S'$ is co-minimal in $\mathscr{D}$, FALSE otherwise
if $\mathsf{C}'_{*,|S'|}$ has zero-entries then                /* zero-coefficients?  */
    return FALSE ;                                           /* subset is not co-min */
else
    return TRUE ;                                                /* subset is co-min */
```

Algorithm 19: Check whether a subset of columns is contained in a circuit utilizing the local data given by a column representation matrix

A	# symmetries	# circuits up to symmetry	# circuits in total	# nodes	CPU time [hh:mm:ss]
C^2	1	1	10	8	0:00:00
C^3	48	3	20	30	0:00:00
C^4	384	15	1348	218	0:00:00
C^5	3840	186	353,616	2615	0:00:00
$*C^6$	46,080	12,628	446,148,992	119,637	0:00:01
$*C^7$	645,120	3,591,868	2,118,502,178,496	25,274,903	0:04:49
$*C^8$	10,321,920	3,858,105,362	38,636,185,528,212,416	21,028,416,820	163:37:00
$*\Delta(8,3)$	40,320	7,240	251,651,820	153,428	0:00:01
$*\Delta(8,4)$	40,320	82,456	3,134,451,775	1,334,075	0:00:09
$*\Delta(9,3)$	362,880	228,432	75,267,509,940	4,298,973	0:00:28
$*\Delta(9,4)$	362,880	31,671,609	11,259,090,122,490	346,869,277	1:05:40
$*\Delta(10,3)$	3,628,800	7,494,056	25,290,095,161,170	140,451,527	0:20:42
$*\Delta(10,4)$	3,628,800	12,609,824,635	45,270,853,845,998,550	110,076,768,815	523:25:52

Table 3: Computational results for the enumeration of circuits in hypercubes and hypersimplices using 16 threads; the results for C^6, C^7, C^8, and the results for hypersimplices are new (marked with a "*"), to the best of my knowledge

10 Application III: Triangulations

Most general-purpose enumeration algorithms for triangulations rely on an algorithm based on the flip graph of triangulations [14, 5, 7]. Since Santos found a triangulation without flips [17] it is known that one might not find all triangulations this way. There have always been hints in the literature on how to enumerate all triangulations of a point configuration by enumerating maximal cliques in the proper-intersection graph of all simplices. However, to date no implementation of this idea could ever compete with flip-based algorithms.

Here, an all new attempt is presented based on symmetric lexicographic subset reverse search with pruning (Algorithm 10). In the following, a subset of pairwise properly intersecting simplices is called a *partial triangulation*.

The single most important ingredient is a very fast semi-check for the non-right-extendability of a subset to a feasible subset, i.e., the non-right-extendability of a partial triangulation to a triangulation.

It is appropriate to use a semi-check, since the decision problem of whether or not a partial triangulation can be extended to a triangulation is NP-hard in general. This can be derived from the NP-hardness of triangulating non-convex 3-polytopes [16], because for a general partial triangulation the uncovered part yet to be triangulated is, in general, non-convex. Here, only extensions to the right are interesting w.r.t. a certain ordering of simplices. It would be interesting to try to formally extend the NP-hardness result to this restricted case.

As a first step towards the application of Algorithm 10, one has to specify how triangulations are represented as subsets of some finite set. To this end, let $\mathbf{A}$ be a rank-r configuration with n elements. Let $\mathscr{S}$ be the set of independent r-subsets of column indices of $\mathbf{A}$ (called *simplices*) and n_s be its cardinality. Similarly, let $\mathscr{F}$ be the set of all independent $r-1$-subsets of column indices of $\mathbf{A}$ (called *simplex-facets*, or simply *facets* whenever no confusion with the facets of $\mathbf{A}$ arises) and n_f be its cardinality.

By fixing an arbitrary bijection $\mathscr{S} \to [n_s]$ to encode simplices, any triangulation $\mathscr{T}$ given by its set of maximal simplices can be represented as a subset T of $[n_s]$. While any bijection will allow to use Algorithm 10, there is a special bijection that helps to accelerate the semi-check for the non-right-extendability of a subset. Here, the lexicographic ordering plays an important role. For the rest of this section, let $\mathrm{idx_s}\colon \mathscr{S} \to [n_s]$ be the bijection that assigns to each simplex its position in the lexicographic order of simplices. In other words: choose the unique bijection preserving the respective natural total order of elements in each set. Similarly, let $\mathrm{idx_f}\colon \mathscr{F} \to [n_f]$ be the bijection that assigns to each simplex-facet its position in the lexicographic order of facets. For convenience, denote the inverse functions by $\mathrm{simp} = \mathrm{idx_s}^{-1}$ and $\mathrm{facet} = \mathrm{idx_f}^{-1}$, respectively. With this notation, the subset T corresponding to a triangulation $\mathscr{T}$ is given by $T = \{s \in [n_s] : \mathrm{simp}(s) \in \mathscr{T}\}$, and the triangulation corresponding to a subset T is given by $\mathscr{T} = \{S \in \mathscr{S} : \mathrm{idx_s}(S) \in T\}$. In other words, T *indexes* $\mathscr{T}$.

A brief inspection of Algorithm 10 shows that this results in the following: not only are the triangulations enumerated in the lexicographic order of $2^{[n_s]}$, but also each triangulation itself is built by adding simplices one-by-one in lexicographic order.

The downset $\mathscr{D}$ used in the method is the set of subsets indexing simplex subsets that are pairwise properly intersecting. The set $\mathscr{F}$ of feasible subsets is the set of all subsets indexing a triangulation. Since no triangulation contains any other triangulation, one can apply Algorithm 10 by specializing its subroutines `IsFeasible`, `Expand`, and `SemiIsNotRightExt`.

Next, some notions are introduced that help to set up appropriate global and local auxiliary data, which will speed up the implementation. The global auxiliary data is defined first. It is desirable to access quickly the incidence information of spanning simplices and their facets. Given the characterization of a triangulation in Definition 10, the distinction between interior and boundary simplex-facets is important.

Definition 12. For a rank-r-configuration $\mathbf{A}$ with n points, n_s many simplices, and n_f many simplex-facets, define the *interior-facets table* as

$$\text{intfacets}\colon \begin{cases} [n_s] & \longrightarrow & 2^{[n_f]}; \\ s & \longmapsto & \big\{f \in [n_f] : \text{facet}(f) \text{ is an interior facet of simp}(s)\big\}, \end{cases} \tag{14}$$

and the *boundary-facets table* as

$$\text{bndfacets}\colon \begin{cases} [n_s] & \longrightarrow & 2^{[n_f]}; \\ s & \longmapsto & \big\{f \in [n_f] : \text{facet}(f) \text{ is a non-interior facet of simp}(s)\big\}. \end{cases} \tag{15}$$

In Algorithm 10 an expansion sequence is used. In the current application such a sequence corresponds to a sequence of lexicographically greater simplices that can be added to a partial triangulation without violating proper intersection. Such a sequence can be updated more easily when for all simplices the set of all simplices that have a proper intersection with it can be accessed quickly.

Definition 13. For a rank-r-configuration $\mathbf{A}$ with n points and n_s many simplices, define the *admissibles table* as

$$\text{admsimps}\colon \begin{cases} [n_s] & \longrightarrow & 2^{[n_s]}; \\ s & \longmapsto & \big\{s' \in [n_s] : \text{simp}(s') \text{ intersects properly with simp}(s)\big\}. \end{cases} \tag{16}$$

Next, some local auxiliary data is defined: with each subset indexing a partial triangulation store

- the index set of all lex-greater simplices intersecting properly with it;
- the index set of all uncovered interior facets.

To this end, define:

Definition 14. Let T index a partial triangulation. Then the *admissibles of T* are defined as

$$\text{admissibles}(T) := \big\{s \in [n_s] : s > s' \text{ and simp}(s) \text{ intersects properly with simp}(s') \text{ for all } s' \in T\big\}. \tag{17}$$

Moreover, define the *free interior facets of T as*

$$\text{freefacets}(T) := \big\{f \in [n_f] : \text{facet}(f) \text{ is an interior facet of simp}(s) \text{ for exactly one } s \in T\big\}. \tag{18}$$

From these data, some useful information can directly be derived.

Lemma 7. *Let T index a non-empty partial triangulation. Then:*

(i) T indexes a triangulation if and only if $\text{freefacets}(T) = \emptyset$.

(ii) If $\text{admissibles}(T) = \emptyset$ *and* $\text{freefacets}(T) \neq \emptyset$*, then T is not right-extendable.*

(iii) The admissibles of an expanded partial triangulation can be computed as

$$\text{admissibles}(T \cup \{s\}) = \{s' \in \text{admissibles}(T) : s' > s\} \cap \text{admsimps}(s). \tag{19}$$

(iv) The free interior facets of an expanded partial triangulation can be computed as

$$\text{freefacets}(T \cup \{s\}) = \text{freefacets}(T) \bigtriangleup \text{intfacets}(s). \tag{20}$$

Proof. The assertions follow from straight-forward checks of definitions. □

This is essentially what was used to-date in any attempt to enumerate all triangulations of a configuration, compare [14, 5]. These methods did not scale well because without any additional pruning they would process a very large number of nodes. For later reference, call this the *no-pruning* method.

One significant improvement over this is the following necessary condition for expansions being extensions.

Theorem 3. *Let T index a non-empty partial triangulation with* $\operatorname{freefacets}(T) \neq \emptyset$ *and* $\operatorname{admissibles}(T) \neq \emptyset$.

Then, if $T \cup \{s'\}$ is right-extendable for some $s' \geq s$ with $s, s' \in$ $\operatorname{admissibles}(T)$*, then*

$$\min\bigl(\operatorname{freefacets}(T)\bigr) \geq \min\bigl(\operatorname{intfacets}(s)\bigr). \tag{21}$$

Proof. If the minimal interior facet index of an admissible s is too large to cover the minimal free facet of T, then the same holds for each admissible $s' > s$ as well, since

$$\min\bigl(\operatorname{intfacets}(s')\bigr) \geq \min\bigl(\operatorname{intfacets}(s)\bigr) \text{ for each } s' > s. \tag{22}$$

The theorem is a formal version of this observation. □

Since all partial triangulations are built in lexicographic order, Theorem 3 means the following: the loop over the expansion sequence in Algorithm 10 can be left as soon as the minimal interior facet index of the new simplex index is larger than the minimal free interior facet index of T, in which case $T \cup \{s\}$ cannot be right-extended. The lexicographic building order guarantees that the minimal free interior facet cannot be covered by any simplex added in the future either. The application of this necessary condition is done inside the `Expand` subroutine and is called *lex-breaking*.

The use of the following quite straight-forward necessary condition for right-extendability additionally prunes the search tree effectively in all experiments at the cost of a substantial computational effort. It was this necessary condition that, for the first time, allowed the enumeration of all symmetry classes of triangulations for instances like the 4-cube (a standard benchmark that has 247,451 symmetry classes of triangulations) in a CPU time comparable to the CPU times of flip-based algorithms.

Theorem 4. *Let T index a non-empty partial triangulation. Then: If T is right-extendable, then there is a covering set of simplex indices $C \subseteq$* $\operatorname{admissibles}(T)$ *such that*

(i) the covering set is pairwise properly intersecting: For each $s \in C$ one has for all $s' \in C \setminus \{s\}$ that $s \in \operatorname{admsimps}(s')$,

(ii) the covering set covers all free interior facets: For each $f \in$ $\operatorname{freefacets}(T)$ *there is an $s \in C$ with* $f \in \operatorname{intfacets}(s)$.

Proof. Note that any right-extension of T to a triangulation must

- restrict to the current admissibles of T,
- is itself properly intersecting,
- contain for each free interior facet a simplex containing that facet.

Thus, any right-extension is a special case of a cover set C as in the theorem. □

Call the application of Theorem 4 *full-pruning.* Note that the existence of a cover set as in Theorem 4 does not guarantee the right-extendability, since its elements may lead to new free interior facets that have to be covered by even more simplices.

Experiments show that full-pruning can be implemented reasonably fast and reduces the number of visited nodes drastically. However, given the lex-orders of simplices and facets, one can prove another necessary condition for right-extendability that can be evaluated much faster and is – surprisingly – almost as effective. It is based on the following theorem that is very similar to Theorem 3.

Theorem 5. *Let T index a non-empty partial triangulation with* $\operatorname{freefacets}(T) \neq \emptyset$ *and* $\operatorname{admissibles}(T) \neq \emptyset$.

Then, if T is right-extendable, we have

$$\min\bigl(\operatorname{freefacets}(T)\bigr) \geq \min\Bigl(\operatorname{intfacets}\bigl(\min(\operatorname{admissibles}(T))\bigr)\Bigr). \tag{23}$$

Proof. If T is right-extendable, then for each free facet indexed in $\operatorname{freefacets}(T)$ there must exist an admissible simplex indexed in the admissibles of T covering it, since $\operatorname{admissibles}(T') \subseteq \operatorname{admissibles}(T)$ for all $T' \supseteq T$. In particular, for the lex-minimal free facet there must be such an admissible simplex. The lex-minimal facet that can be covered by some admissible simplex is the lex-minimal facet of the lex-minimal admissible simplex. Thus, if the lex-minimal free facet were lex-smaller than this, then it could not be covered by *any* admissible simplex of any superset $T' \supseteq T$, and T would not be right-extendable. The first assertion is just a translation of this into a formula. □

Call the application of Theorem 5 *lex-pruning.* In Table 4 a comparison of the node counts is presented for no-pruning and no lex-breaking, full-pruning with lex-breaking, and lex-pruning with lex-breaking for tiny to small examples. No-pruning is not competitive, which explains the limited success of all implementations previously available. Moreover, the examples (and others not in the table) exhibit that full-pruning is slightly more effective than lex-pruning. However, the nodes pruned extra do not justify the substantially larger effort.

Now, the specialized subroutines can be presented for application of Algorithm 10 to the enumeration of *all* triangulations. This time, the global and local auxiliary data are more voluminous. The global data **G** consists of the configuration **A** with hash maps AT for its admissibles and IT for its interior facets. The local data **L** stored with each subset T of simplex indices consists, besides the local data necessary for the lex-min check (in this case a critical-element table CET of T), a set ADM representing the admissible simplices of T and a set FIF representing the free interior facets of T.

The feasibility check just needs part of the local data and is straight-forward (see Algorithm 20).

```
Algorithm: IsFeasible($T$,$\mathscr{F}$,FIF)
Input: A subset $T$ of simplex indices in $[n_s]$, a set of feasible subsets $\mathscr{F}$ (implicit), the set FIF of free
       interior facets of $T$
Output: TRUE if $T$ is a triangulation, FALSE otherwise
if FIF $= \emptyset$ then                                /* no free interior facets?  */
    return TRUE ;                            /* partial triangulation is covering */
else
    return FALSE ;                       /* partial triangulation is not covering */
```

Algorithm 20: Check whether a partial triangulation is a triangulation by checking whether all free interior facets are covered

When processing a node, the corresponding partial triangulation is expanded by a simplex admissible for it. Thus, the set ADM in the local data of a node and ordered in the natural way yields

A	# triangulations in total	# nodes by pruning method no	full	lex	CPU time [hh:mm:ss] by pruning method no	full	lex
C^3	74	2914	485	496	0.02	0.01	0.01
$\Delta_3 \times \Delta_2$	4488	2,385,960	29,422	29,576	1.31	0.11	0.06
$C(9,4)$	357	8,627,256	4860	4925	12.87	0.08	0.02
$C(10,4)$	4824	>5,180,000,000	73,084	73,258	>29:54:40.79	0.83	0.11
$C(11,4)$	96,426	–	1,597,365	1,597,783	–	20.16	1.72

Table 4: Comparison of the three pruning methods no-pruning, full-pruning, and lex-pruning for a 3-cube, a product of a tetrahedron and a triangle, and some cyclic polytopes (tiny to small instances); symmetry handling has been deactivated, and only a single thread has been used in order to focus on the effectivity and efficiency of pruning alone; the number of nodes for no-pruning is too large to handle even small-sized examples; full-pruning is most effective but slower than lex-pruning, which is almost as effective and can be implemented very fast; note that the no-pruning computation for $C(10,4)$ was interrupted after the given time with the given number of nodes; cyclic polytopes pose a stress test for building triangulations simplex-by-simplex because they tend to have many simplices and many triangulations: $C(8,3)$ has 5 through 15 simplices in its 138 triangulations whereas C^3 with the same number of points in the same rank has 5 through 6 simplices in its 74 triangulations, and while the more than 92 million total triangulations of C^4 constitute a small example by today's standards, even without exploiting symmetries, the number of triangulations of $C(16,4)$ is not even known to date, and even $C(15,4)$ with one point less has already more than 565 billion triangulations, see below for exact counts

an expansion sequence. When a new simplex is added, one can generate the new local data from the local data of T, as is described in Algorithm 21. Note that lex-breaking according to Theorem 5 might be possible.

Finally, Algorithm 22 lists the pruning step that reveals in many cases when a partial triangulation can certainly not be right-extended to a triangulation.

This section is closed by Table 5 with some results obtained by the application of the resulting algorithm `SymLexSubsetRSFeas_withData` to the enumeration of all triangulations of point configurations. The known numbers could be computed much faster than in previous experiments (see, e.g., [7]), and some numbers have never been computed before, to the best of my knowledge. The largest new examples are the number of triangulations of the pyritohedron (largest number up to symmetry) and the number of triangulations of $\Delta_5 \times \Delta_3$ (largest total number).

We close this section with a remark on optimization.

Remark 1. Any variant of algorithm `SymLexSubsetRS` – as any enumeration algorithm – can be turned into a branch-and-bound optimization algorithm by specifying an objective function and a dual-bound procedure. We have used this to compute the minimal number of simplices in a triangulation of some examples. We will not go into the details, since the method we implemented is straight-forward. Example results are: a minimal triangulation of the product of a square and a triangle has 10 simplices (a result originally proved in [18] with quite some effort), which took less than a tenth of a second; a minimal triangulation of the 4-cube has 16 simplices, which took less than half a second; a minimal triangulation of the regular dodecahedron has 23 simplices, which took less than 10 minutes; a minimal triangulation of the pyritohedron has 23 simplices, too, which took less than 4 hours. All computation times for optimization are significantly shorter than those for the complete enumeration.

```
Algorithm: Expand($T, s, \mathbf{G}, \mathbf{L}$)
Input: A subset $T$ of simplex indices in $[n_s]$, a new simplex index $s$, global data $\mathbf{G}$ encompassing the
       interior-facets table IT and the admissibles table AT of $\mathbf{A}$, local data $\mathbf{L}$ of $T$ encompassing the
       free interior facets FIF and the admissibles ADM of $T$
Output: (break, $T', \mathbf{L}'$), where break is true if this and all future expansions cannot be extensions,
        $(T', \mathbf{L}')$ is the new node with $T' = T \cup \{s\}$, and $\mathbf{L}'$ is the correct local Data for $T'$
if $\min(\mathsf{FIF}) < \min(\mathsf{IT}(s))$ then          /* can $s$ possibly cover minimal free facet? */
    return (TRUE, –, –);                         /* min. free facet not coverable by $s' \geq s$ */
$T' \leftarrow T \cup \{s\}$;                                              /* add $s$ to the subset */
$\mathsf{FIF}' \leftarrow \mathsf{FIF} \triangle \mathsf{IT}[s]$;                              /* free interior facets by x-oring */
$\mathsf{ADM}' \leftarrow \{s' \in \mathsf{ADM} : s' > s\} \cap \mathsf{AT}[s]$;          /* admissibles t.t.r. by intersection */
$\mathbf{L}' \leftarrow (\mathsf{FIF}', \mathsf{ADM}')$;                                    /* put together local data */
return (FALSE, $T', \mathbf{L}'$);                                        /* return the node */
```

Algorithm 21: Expand a subset indexing a partial triangulation by a new simplex index; at the same time compute its new local data consisting of free interior facets and admissible simplices

```
Algorithm: SemiIsNotRightExt($T', \mathscr{D}, \mathscr{F}, \mathbf{G}, \mathbf{L}, \mathbf{L}'$)
Input: A subset $T'$ of simplex indices in $[n_s]$, the downset $\mathscr{D}$ of subsets indexing partial triangulations,
       the set $\mathscr{F}$ of triangulations (given implicitly by IsFeasible), global data $\mathbf{G}$ encompassing the
       interior-facets table IT and the admissibles table AT of $\mathbf{A}$, local data $\mathbf{L}$ of $T$ encompassing the
       free interior facets FIF and the admissibles ADM of $T$
Output: TRUE if $T'$ is not right-extendable, FALSE if $T'$ may be right-extendable
if $\min(\mathsf{FIF}) < \min(\mathsf{IT}[\min(\mathsf{ADM})])$ then            /* min. free facet not coverable? */
    return TRUE;                        /* min. free facet not coverable by admissibles */
else
    return FALSE;                             /* no contradiction to right-extendability */
```

Algorithm 22: Check whether a partial triangulation can certainly not be right-extended to a triangulation by direct application of Theorem 5

A	# symmetries	# triangulations up to symmetry	# triangulations in total	# nodes	CPU time [hh:mm:ss]
C^4	384	247,451	92,487,256	3,446,658	0:00:02
$\Delta_6 \times \Delta_2$	30,240	533,242	16,119,956,160	6,325,471	0:03:45
$\Delta_4 \times \Delta_3$	2880	7,402,421	21,316,106,880	116,083,389	0:05:12
*$\Delta_5 \times \Delta_3$ (M1Max)	17280	25,606,173,722	442,472,050,753,920	429,725,338,123	1313:57:17
*$\Delta(7,2)$	5040	37,676,752	189,355,661,460	10,222,438,919	1:39:26
*$\Delta(6,3)$	720	119,186,888	85,793,497,200	38,860,274,741	1:00:03
$3\Delta_3$	24	925,148,763	22,201,684,367	7,154,329,211	0:17:52
$C(14,8)$ (old)	28	2,429,751	68,007,706	187,209,581	0:00:36
$C(19,14)$ (old)	38	6,515,385	247,567,074	1,265,333,659	0:07:07
*$C(16,3)$ (old)	2	58,492,955,941	116,985,744,912	887,659,233,812	30:44:20
*$C(14,4)$ (old)	28	244,771,183	6,853,476,616	8,343,040,543	0:23:07
*$C(15,4)$ (old)	30	18,845,509,142	565,365,033,880	650,520,069,379	34:05:57
*$C(14,5)$ (old)	2	328,152,636,588	656,305,030,644	11,575,302,270,564	475:34:44
*$C(14,6)$ (old)	28	8,314,337,199	232,797,963,456	535,970,897,964	30:27:47
*$C(14,7)$ (old)	2	15,813,939,113	31,627,843,174	937,148,113,629	40:44:32
*$C(15,9)$ (old)	2	1,397,895,884	2,795,741,709	117,124,014,922	6:04:27
*$C(16,10)$ (old)	32	1,902,605,255	60,881,310,552	213,053,994,783	16:45:55
*icosahedron	120	95	8598	3812	0:00:00
*pseudoicosahedron	24	7701	182,670	147,774	0:00:00
*dodecahedron	120	12,775,757,027	1,533,079,037,570	125,333,463,448	8:28:49
*pyritohedron	24	1,363,918,758,719	32,734,029,351,118	13,786,801,148,393	607:18:52

Table 5: Computational results for the enumeration of triangulations using 16 threads; the results for $\Delta_5 \times \Delta_3$, the hypersimplices, some cyclic polytopes, as well as the regular icosahedron, the pseudoicosahedron (a fibonacci approximation of it), the regular dodecahedron, and the pyritohedron (a fibonacci approximation of it) are new, to the best of my knowledge (marked with a "*"); the CPU times a far smaller throughout than for any earlier effort

11 Conclusions

Variants of the generic algorithm SymLexSubsetRS have been introduced to enumerate maximal, co-minimal, and feasible subsets of a finite sets up to symmetry. New versions of lex-minimality checks and new pruning methods for partial cocircuits and partial triangulations have been presented. Interestingly enough, no non-trivial pruning-method could be found for subsets that are not contained in any circuit. The new methods allowed for the computation of many new cardinalities, among them the number, up to symmetry, of cocircuits of the 9-cube, the number of circuits of the 8-cube, and the number of all triangulations of the dodecahedron.

The algorithm SymLexSubsetRS can be enhanced by an objective function and a dual-bound procedure to generate a straight-forward optimization algorithm. This was applied to find triangulations with a minimal number of simplices for the 4-cube and the regular dodecahedron in much less time than the enumeration takes. Since this was only used in a straight-forward manner, no details were explained to this end. The efficiency of such an optimization algorithm will depend on the quality of the dual-bound procedure. It would be interesting to see where such an approach would be competitive to other special optimization algorithms (like the universal-polytope approach [10] for the minimal triangulation).

The new methods should have many more applications beyond the three applications in this paper like the enumeration of maximal cliques in a graph up to symmetry. This will be subject of further research.

12 Acknowledgement

The author is very grateful to Michael Joswig and Lars Kastner who provided the exact chirotope of the regular dodecahedron, computed by polymake. Chirotope data makes possible the exact computation of all triangulations of a point configurations even if there are no rational coordinates for it. Moreover, the author thanks Lisa Lamberti and Komei Fukuda for communicating the (at that time open) question about the number of circuits of the 6-cube, which led to the simultaneous consideration of triangulations and circuits, which was beneficial for both applications.

References

[1] Oswin Aichholzer and Franz Aurenhammer. Classifying hyperplanes in hypercubes. *SIAM Journal on Discrete Mathematics*, 9(2):225–232, 1996.

[2] David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1):21 – 46, 1996. First International Colloquium on Graphs and Optimization.

[3] David Avis and Charles Jordan. mts: a light framework for parallelizing tree search codes. *Optimization Methods and Software*, 0(0):1–22, 2019.

[4] Jesús A. de Loera. *Triangulations of Polytopes and Computational Algebra*. PhD thesis, Cornell University, 1995.

[5] Jesús A. de Loera, Jörg Rambau, and Francisco Santos. *Triangulations – Structures for Applications and Algorithms*, volume 25 of *Algorithms and Computation in Mathematics*. Springer, 2010.

[6] Hiroshi Imai, Tomonari Masada, Fumihiko Takeuchi, and Keiko Imai. Enumerating triangulations in general dimensions. *International Journal of Computational Geometry & Applications*, 12(06):455–480, 2002.

[7] Charles Jordan, Michael Joswig, and Lars Kastner. Parallel enumeration of triangulations. *The electronic journal of combinatorics*, 25(3):P3.6, 2018.

[8] Michael Joswig and Lars Kastner. New counts for the number of triangulations of cyclic polytopes. In James H. Davenport, Manuel Kauers, George Labahn, and Josef Urban, editors, *Mathematical Software – ICMS 2018*, pages 264–271, Cham, 2018. Springer International Publishing.

[9] L. Khachiyan, E. Boros, K. Elbassioni, V. Gurvich, and K. Makino. On the complexity of some enumeration problems for matroids. *SIAM Journal on Discrete Mathematics*, 19(4):966–984, 2005.

[10] Jesús A. de Loera, Serkan Hoşten, Francisco Santos, and Bernd Sturmfels. The polytope of all triangulations of a point configuration. *Documenta Mathematika*, 1:103–119, 1996.

[11] Arnaud Mary and Yann Strozecki. Efficient enumeration of solutions produced by closure operations. *CoRR*, abs/1712.03714, 2019.

[12] E. Minieka. Finding the circuits of a matroid. *Journal of Research of the National Bureau of Standards, Section B: Mathematical Sciences*, page 337, 1976.

[13] Christian Pech and Sven Reichard. Enumerating set orbits. In Mikhail Klin, Gareth A. Jones, Aleksandar Jurišić, Mikhail Muzychuk, and Ilia Ponomarenko, editors, *Algorithmic Algebraic Combinatorics and Gröbner Bases*, pages 137–150. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[14] Jörg Rambau. TOPCOM: Triangulations of point configurations and oriented matroids. In *Proceedings of the International Congress of Mathematical Software*, 2002.

[15] Jörg Rambau and Victor Reiner. A survey of the higher stasheff-tamari orders. In Folkert Müller-Hoissen, Jean Marcel Pallo, and Jim Stasheff, editors, *Associahedra, Tamari Lattices and Related Structures – Tamari Memorial Festschrift*, volume 299 of *Progess in Mathematics*, chapter 18, pages 351–390. Springer, 2012.

[16] Jim Ruppert and Raimund Seidel. On the difficulty of triangulating three-dimensional nonconvex polyhedra. *Discrete & Computational Geometry*, 7:227–253, 1992.

[17] Francisco Santos. A point set whose space of triangulations is disconnected. *Journal of the American Mathematical Society*, 13:611–637, 2000.

[18] Tyler Seacrest and Francis Edward Su. A lower bound technique for triangulations of simplotopes. *SIAM Journal on Discrete Mathematics*, 32(1):1–28, 2018.